

Tesina de Grado
para la obtención del título de
Licenciado en Ciencias de la Computación

Seguridad en iOS y Android: un Análisis Comparativo

Autor

Raúl Ignacio Galuppo

raul.i.galuppo@gmail.com

G-3483/5

Director

Dr. Carlos Luna

Facultad de Ingeniería

Universidad de la República

Uruguay



Departamento de Ciencias de la Computación
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario
Argentina

Diciembre de 2017

Índice general

1. Introducción	1
2. Modelo de Seguridad de Android	3
2.1. Entorno aislado para cada aplicación	3
2.2. Políticas de acceso mejoradas	4
2.3. Seguridad en las aplicaciones	5
2.3.1. Permisos	5
2.3.2. Manifiesto	6
3. Modelo de Seguridad de iOS	8
3.1. Protección de los datos	8
3.2. Seguridad en las aplicaciones	10
3.2.1. Entorno seguro	10
3.2.2. Controles de privacidad	11
4. Análisis Comparativo	12
4.1. Analizando Android	13
4.1.1. Autenticación del usuario	13
4.1.2. Seguro desde el arranque	14
4.1.3. Cifrado de la partición de datos	16
4.1.4. Permisos	17
4.2. Analizando iOS	18
4.2.1. Bloqueo del dispositivo	18
4.2.2. Arranque seguro	19
4.2.3. Cifrado de archivos	20
4.2.4. Firma de código de las aplicaciones	21
4.2.5. Permisos	21
4.3. Crítica	23
4.3.1. Arranque verificado	23
4.3.2. Cifrado del sistema de archivos	24
4.3.3. Bloqueo del dispositivo	24
4.3.4. Seguridad de las aplicaciones	25

5. Apache Cordova	28
5.0.5. Arquitectura	28
5.0.6. Aplicaciones Nativas vs Aplicaciones Híbridas	29
5.0.7. Plugins	30
6. Hacia un Framework Comparativo	31
6.1. Ejecución de los test	31
6.1.1. Emulador oficial de Android	32
6.1.2. Simulador oficial de iOS	32
6.2. Vista principal	34
6.2.1. Funciones no compatibles	35
6.3. Catálogo de Tests	36
6.3.1. Contactos	36
6.3.2. Calendario	37
6.3.3. Geolocalización	38
6.3.4. SMS	40
6.3.5. Almacenamiento	41
6.3.6. Información del Dispositivo	41
6.3.7. Sensores	42
6.3.8. Internet	43
6.4. Resultados experimentales	44
6.4.1. Clase A	45
6.4.2. Clase B	46
6.4.3. Clase C	46
6.4.4. Clase D	47
7. Conclusiones y Trabajos Futuros	49
Anexo	52
Bibliografía	58

Índice de figuras

2.1.	Aislamiento de las aplicaciones según su UID [6].	4
2.2.	Los recursos están resguardados por los permisos [5].	5
2.3.	Manifiesto de una aplicación.	7
3.1.	Modelo de seguridad de iOS [10].	9
3.2.	Entorno aislado de una aplicación en iOS [11]	10
3.3.	Control de privacidad de iOS 9.	11
4.1.	Proceso de autenticación en Android[5].	14
4.2.	Diagrama de flujo del <i>Arranque seguro</i> [6].	15
4.3.	Captura de Ajustes/Seguridad	16
4.4.	Permisos en Android Marshmallow.	19
4.5.	Proceso para descifrar un archivo [10].	20
4.6.	Permisos en iOS 9.	22
5.1.	Arquitectura de Apache Cordova [16].	29
6.1.	Configuración de un AVD.	33
6.2.	Pantalla de inicio del simulador de iOS [12].	34
6.3.	Áreas del <i>framework</i>	35
6.4.	Testeando la administración de los contactos.	37
6.5.	Testeando la administración del calendario.	38
6.6.	Testeando la geolocalización.	39
6.7.	Panel de configuración del emulador de Android.	39
6.8.	Testeando los mensajes SMS.	40
6.9.	Testeando el almacenamiento del dispositivo.	41
6.10.	Testeando Información del Dispositivo.	42
6.11.	Testeando los sensores.	43
6.12.	Testeando el acceso a Internet.	44
1.	Estructura de archivos de un proyecto Apache Cordova.	53
2.	Estructura típica de una aplicación híbrida.	54

Índice de cuadros

4.1. Comparación de permisos entre Android 6.0 e iOS	26
6.1. Clasificación de permisos según si requieren autorización. . .	45

Índice de Algoritmos

1.	Test de Contactos.	36
2.	Test del Calendario.	37
3.	Test de Geolocalización.	38
4.	Test de SMS.	40
5.	Test de Almacenamiento.	41
6.	Test de Información del Dispositivo.	42
7.	Test de los Sensores.	42
8.	Test de conexión a Internet.	43

Resumen

La seguridad en dispositivos móviles se ha convertido en un asunto muy importante debido al incremento de ataques recibidos y a las consecuencias que éstos tienen. Los ataques están incentivados por la popularización de los dispositivos móviles, el aumento de información personal y confidencial que almacenan y las operaciones realizadas a través de ellos, como por ejemplo las bancarias. Consecuentemente, es de vital importancia analizar sus respectivos modelos de seguridad, con el objetivo de encontrar similitudes y diferencias, así como fortalezas y debilidades de cada uno.

Este trabajo realiza un análisis detallado sobre las características de seguridad en Android e iOS, para sus versiones más avanzadas, con el objetivo de preservar la privacidad del usuario. En particular, considerando permisos que se pueden modificar en tiempo de ejecución. Sumado al análisis, se presenta un *framework* comparativo. Es una aplicación móvil híbrida, que puede ejecutarse tanto en Android como en iOS. Tiene dos funciones principales: determinar empíricamente los alcances de los sistemas de permisos; y establecer una relación entre los permisos presentes en ambas plataformas. El *framework*, que puede ser extendido, pone especial énfasis en la relación existente entre la privacidad del usuario y el sistema de permisos.

Capítulo 1

Introducción

En los últimos años se ha observado un marcado incremento en el número de dispositivos móviles que tienen a iOS y Android como sistemas operativos. Actualmente, más del 99 % de los dispositivos móviles en el mercado tiene uno de estos sistemas operativos [18]. Hay muchas aplicaciones que se desarrollan para estas dos plataformas. El número actual de aplicaciones de Android en el mercado supera los 3.500.00 [9] y para iOS asciende a más de 3.100.000 [23]. Debido al uso diario de estas aplicaciones, se puede filtrar una gran cantidad de información privada y confidencial a menos que se aplique control de acceso a las aplicaciones instaladas.

Android [3] es un sistema operativo de código abierto [4], diseñado para dispositivos móviles y desarrollado por Google junto con la Open Handset Alliance [1]. Su modelo de seguridad, a pesar de poseer características muy variadas, presenta ciertas limitaciones. Algunos trabajos previos dan cuenta de la rigidez del sistema de permisos a la hora de, por ejemplo, instalar una nueva aplicación. Además, muchos de los aspectos de la seguridad en Android dependen de la correcta construcción de las aplicaciones por parte de los desarrolladores y, al mismo tiempo, no existe una documentación precisa que facilite dicha tarea.

Por otra parte, iOS es un sistema operativo móvil de la multinacional Apple Inc. diseñado para ser seguro [10]. Cada dispositivo combina *hardware*, *software* y servicios, diseñados para trabajar conjuntamente para proveer seguridad y al mismo tiempo, que la misma sea transparente para el usuario. Las principales características de seguridad, como el cifrado del dispositivo, no son configurables y vienen habilitadas por defecto, por lo que los usuarios no pueden deshabilitarlas por error. La seguridad se extiende más allá del dispositivo, incluido todo lo que hacen los usuarios localmente, en redes y con servicios clave de Internet. Como consecuencia de ello, se

genera un ecosistema seguro.

Analizar detalladamente las características de seguridad en ambos sistemas, para sus versiones más avanzadas, permitirá entender las fortalezas y debilidades comparativas de cada uno. Existen muchas formas de comparación posibles. Por ejemplo, la medida propuesta en [28] consiste en analizar la seguridad de una aplicación móvil en cada fase del ciclo de vida, comparándola en cada plataforma. En [22] el enfoque es distinto; se centra en comparar los permisos requeridos a cada plataforma al momento de instalar aplicaciones presentes en ambos sistemas. En [20, 15, 26] se cambia el enfoque propuesto. El objetivo que persiguen estos trabajos es desarrollar una especificación formal que describa el módulo de seguridad de Android. En cambio, en [27], el enfoque se centra en distintos ataques al modelo de seguridad de iOS. En el presente trabajo se analizarán formas alternativas de comparación. En particular, considerando permisos que se pueden modificar en tiempo de ejecución.

A continuación se describe la organización del trabajo. En el segundo y tercer capítulo se presentan los modelos de seguridad de Android e iOS, respectivamente. El análisis comparativo de ambos sistemas operativos se realiza en el cuarto capítulo. Luego, en el quinto capítulo se introduce la plataforma de desarrollo Apache Cordova. Con ella, se realizó un *framework* que permite comparar empíricamente los sistemas de permisos de Android e iOS. Los resultados de dicha comparación, junto con la composición del *framework*, se encuentran en el sexto capítulo. En el séptimo y último capítulo se detallan las conclusiones y los trabajos futuros. Finalmente, se incluye un anexo en donde se encuentran los pasos para descargar el proyecto del *framework* y una guía orientativa para extenderlo.

Capítulo 2

Modelo de Seguridad de Android

Android [3] es un sistema operativo de código abierto, diseñado para dispositivos móviles y desarrollado por Google junto con la Open Handset Alliance. Su arquitectura sigue el estilo arquitectónico conocido como Sistemas Estratificados: los distintos componentes se agrupan en capas según su nivel de abstracción, conformando una jerarquía. Las capas inferiores contienen componentes ligados al *hardware*, mientras que las capas superiores agrupan componentes ligados con tareas de más alto nivel.

Una de las características principales de Android es que cualquier aplicación, ya sea principal o creada por algún desarrollador, puede, al instalarse con las autorizaciones adecuadas, utilizar tanto los recursos/servicios del dispositivo móvil como los ofrecidos por el resto de las aplicaciones instaladas.

A lo largo de este capítulo se presentará una descripción de los principales aspectos del modelo de seguridad de Android, en la versión 6.0 llamada Marshmallow, lanzada en 2015 [2].

2.1. Entorno aislado para cada aplicación

Fue una de las primeras tecnologías de seguridad aplicadas en Android, y tiene mucha importancia en el modelo de seguridad. Consiste en que cada aplicación se ejecuta en un *entorno aislado*¹, forzando a que solo pueda tener acceso irrestricto a sus propios recursos. Por lo tanto, las aplicaciones no pueden interactuar entre ellas y tienen acceso limitado al sistema operativo. A cada aplicación se le asigna un único id de usuario UID y se ejecuta en ese

¹Traducción propuesta para el término *sandbox*.

usuario como un proceso independiente, tal como se observa en la Figura 2.1.

Dado que el *entorno aislado* es a nivel del *kernel*, este modelo de seguridad se extiende al código nativo y a las aplicaciones del sistema operativo, tales como las bibliotecas del sistema operativo y los *frameworks* de las aplicaciones. Esto genera un aislamiento a nivel del *kernel*, ya que todas las políticas que se aplican a usuarios o grupos de usuarios se transfieren a las aplicaciones por tener su UID.

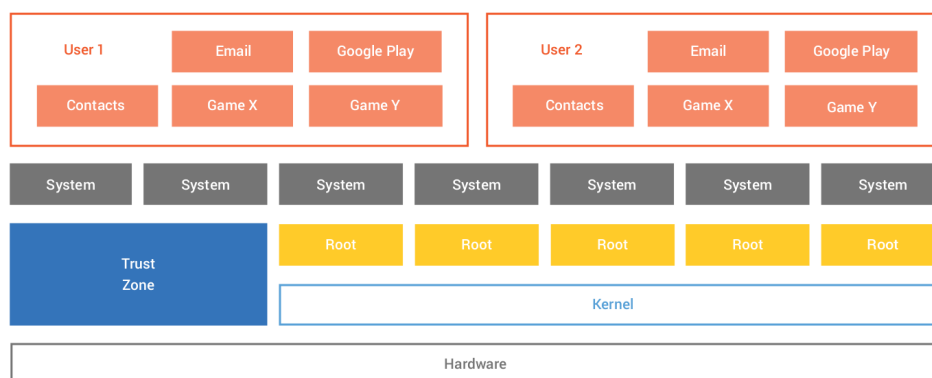


Figura 2.1: Aislamiento de las aplicaciones según su UID [6].

2.2. Políticas de acceso mejoradas

*SELinux*² es un sistema de políticas de acceso obligatorio para Linux. Por lo tanto, siempre se consulta a una autoridad central para permitir cualquier acceso a un recurso; abarca a todos los procesos, inclusive aquellos que corren con privilegios de *root*. *SELinux* opera bajo *the ethos of default denial*, es decir, todo lo que no está explícitamente permitido es denegado.

Decimos que un dominio es una etiqueta que identifica un conjunto de procesos en una política de seguridad. Los procesos que comparten una misma etiqueta de dominio son tratados de la misma forma.

Android asigna un rol, un nombre de usuario y un dominio al usuario. Por lo tanto, aunque varios usuarios pueden compartir el mismo nombre de usuario de , el control de acceso se administra a través del dominio, que está configurado por diferentes políticas. Estas políticas generalmente incluyen instrucciones y permisos específicos, que el usuario debe poseer

²*Security Enhanced Linux*, por sus siglas en inglés.

para obtener acceso al sistema.

SELinux puede trabajar en dos modos:

- Riguroso: en el que los rechazos de permisos se registran y se aplican.
- Permisivo: las denegaciones de permisos se registran pero no se aplican.

Android permite aplicar el modo permisivo en un determinado dominio y el resto del sistema permanece en modo riguroso. Gracias a ello, se puede lograr aplicar incrementalmente *SELinux* a una porción cada vez mayor del sistema y desarrollar de políticas para nuevos servicios, manteniendo el resto del sistema en vigencia.

2.3. Seguridad en las aplicaciones

2.3.1. Permisos

Ciertos recursos que provee Android son sensibles, ya que acceden a datos personales o periféricos importantes. Dichos recursos sólo pueden ser accedidos mediante una SS-API³ con un doble objetivo: tenerlos aislados y permitir cierta granularidad de seguridad sobre ellos [22]. El mecanismo de seguridad para el acceso a estas SS-API de recursos se llama Permisos, tal como se observa en la Figura 2.2.

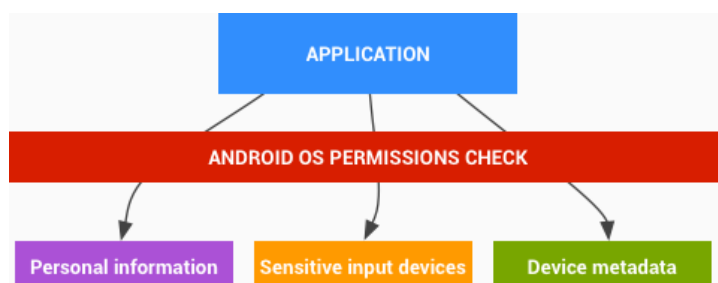


Figura 2.2: Los recursos están resguardados por los permisos [5].

Podemos clasificar los permisos según el riesgo implícito al otorgarlos, resultando las siguientes cuatro categorías [26]:

- *Normal*: Son aquellos permisos de bajo riesgo ya que corresponden a características aisladas y son considerados de bajo riesgo para las demás aplicaciones, para el sistema y para el usuario. Son concedidos automáticamente por el sistema, sin solicitar aprobación explícita del usuario.

³*Security Sensitive API*, por sus siglas en inglés.

- *Dangerous*: Son aquellos permisos de alto riesgo ya que resguardan los accesos a información sensible para el usuario u otorgan control sobre funcionalidades principales del sistema. Para ser concedidos se requiere aprobación explícita del usuario.
- *Signature*: Son aquellos permisos que son aprobados solamente si la aplicación que los requiere tiene el mismo certificado que la aplicación que los creó. Cuando el sistema valida el certificado, se otorga el permiso sin requerir aprobación explícita del usuario. Se crearon para permitir que un desarrollador pueda compartir información entre sus distintas aplicaciones sin necesidad de la aprobación del usuario.
- *Signature/System*: Son aquellos permisos que controlan el acceso a servicios críticos del sistema. En general, las únicas aplicaciones que los utilizan son las que vienen pre-instaladas en el dispositivo, ya que se utilizan para ciertas situaciones especiales en las que varios proveedores tienen aplicaciones integradas en una imagen del sistema que necesitan compartir funciones específicas explícitamente porque se están creando juntas.

2.3.2. Manifiesto

El archivo de manifiesto proporciona información sobre una aplicación al sistema Android, información que el sistema debe tener para poder ejecutar el código de la aplicación. Es por ello que todas las aplicaciones deben tener un archivo **AndroidManifest.xml** (con ese nombre exacto) en el directorio raíz. En este archivo se declaran todos los componentes que forman parte de la aplicación en cuestión, los permisos que son requeridos (ver sección 2.3.1) y los permisos exportados por la aplicación, entre otras cosas. En la Figura 2.3 se observa el manifiesto de una aplicación que requiere los permisos **READ_CONTACTS** y **WRITE_CONTACTS**, entre otros.

```

<?xml version='1.0' encoding='utf-8'?>
<manifest android:hardwareAccelerated="true" android:versionCode="10000"
  android:versionName="1.0.0" package="com.tesina.runtimepermissions"
  xmlns:android="http://schemas.android.com/apk/res/android">
  <supports-screens android:anyDensity="true" android:largeScreens="true"
    android:normalScreens="true" android:resizeable="true"
    android:smallScreens="true" android:xlargeScreens="true" />
  <uses-permission android:name="android.permission.INTERNET" />
  <application android:hardwareAccelerated="true" android:icon="@mipmap/icon"
    android:label="@string/app_name" android:supportsRtl="true">
    <activity android:configChanges="orientation|keyboardHidden|keyboard|
      screenSize|locale" android:label="@string/activity_name"
      android:launchMode="singleTop" android:name="MainActivity"
      android:theme="@android:style/Theme.DeviceDefault.NoActionBar"
      android:windowSoftInputMode="adjustResize">
      <intent-filter android:label="@string/launcher_name">
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <receiver android:name="cordova.plugins.Diagnostic$
      LocationProviderChangedReceiver">
      <intent-filter>
        <action android:name="android.location.PROVIDERS_CHANGED" />
      </intent-filter>
    </receiver>
  </application>
  <uses-sdk android:minSdkVersion="16" android:targetSdkVersion="25" />
  <uses-permission android:name="android.permission.READ.CALENDAR" />
  <uses-permission android:name="android.permission.WRITE.CALENDAR" />
  <uses-permission android:name="android.permission.READ.CONTACTS" />
  <uses-permission android:name="android.permission.WRITE.CONTACTS" />
  <uses-permission android:name="android.permission.GET.ACCOUNTS" />
  <uses-permission android:name="android.permission.ACCESS_COARSE.LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_FINE.LOCATION" />
  <uses-feature android:name="android.hardware.location.gps" />
  <uses-permission android:name="android.permission.SEND.SMS" />
  <uses-permission android:name="android.permission.READ.PHONE.STATE" />
  <uses-feature android:name="android.hardware.telephony" android:required="
    false" />
  <uses-permission android:name="android.permission.WRITE.EXTERNAL.STORAGE" />
</manifest>

```

Figura 2.3: Manifiesto de una aplicación.

Capítulo 3

Modelo de Seguridad de iOS

iOS es un sistema operativo para dispositivos móviles de la multinacional Apple Inc. diseñado para ser seguro. Cada dispositivo combina hardware, software y servicios, diseñados para trabajar conjuntamente para proveer seguridad y al mismo tiempo, que sea transparente para el usuario.

Desde el arranque del dispositivo hasta las actualizaciones de *software* de iOS y las aplicaciones de terceros, cada paso se analiza e investiga para garantizar que el *hardware* y el *software* funcionen de manera óptima y utilicen los recursos correctamente [10]. En la Figura 3.1 se observa los distintos componentes de la arquitectura de iOS. Esta arquitectura es fundamental para la seguridad, y nunca obstaculiza la usabilidad del dispositivo [10].

A lo largo de este capítulo se describirán los principales aspectos del modelo de seguridad de iOS, considerando la versión 9.3, lanzada en 2015.

3.1. Protección de los datos

iOS tiene una protección especial para los archivos y los datos personales, la cual sigue intacta inclusive si algunas otras partes del sistema de seguridad fueron comprometidas [10]. La protección de los datos es implementada construyendo varias claves criptográficas y generando con ellas una jerarquía.

El dispositivo móvil provee un componente de *hardware* llamado *Secure Enclave*. Es un coprocesador con memoria cifrada e incluye generación de números aleatorios por hardware. Tiene tres responsabilidades principales:

- proveer las operaciones de cifrado para la manipulación de la Clave de los Datos¹;

¹Traducción propuesta para el término *Data Protection Key*.

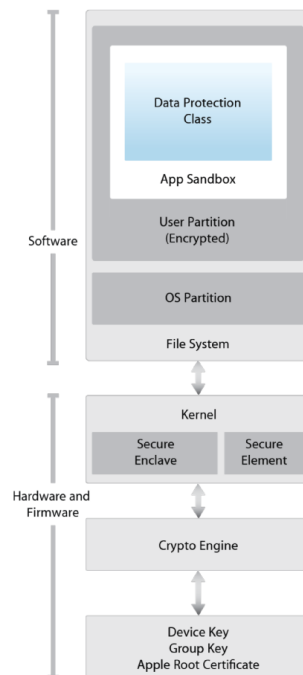


Figura 3.1: Modelo de seguridad de iOS [10].

- asegura la integridad de los datos inclusive si *kernel* fue comprometido;
- es el responsable de procesar los datos provenientes del *Touch ID*², determinando si se desbloquea el dispositivo.

En los procesadores A9 o posteriores de la serie A, el chip genera de forma segura el identificador único UID³. Esta clave fue creada en el momento de fabricación y no es conocida por Apple ni por ningún otro componente del dispositivo. La misma es utilizada para generar una clave efímera cada vez que se prende el dispositivo, la cual se utiliza para cifrar la memoria del *Secure Enclave* y los datos del sistema de archivos.

El UID permite vincular los datos a un dispositivo determinado mediante cifrado. Por ejemplo, la jerarquía de claves que protege el sistema de archivos incluye el UID, de modo que si los chips de memoria se trasladan físicamente de un dispositivo a otro, no será posible acceder a los archivos.

² *Touch ID* es el sistema de detección de huellas digitales que hace posible un acceso seguro, más rápido y sencillo al dispositivo.

³ *Unique ID*, por sus siglas en inglés.

3.2. Seguridad en las aplicaciones

3.2.1. Entorno seguro

Las aplicaciones son un punto crítico en la seguridad de un dispositivo móvil. Es por ello que iOS provee varias capas de seguridad para las aplicaciones, asegurando que una aplicación esté certificada y verificada antes de estar disponibles en la tienda [10].

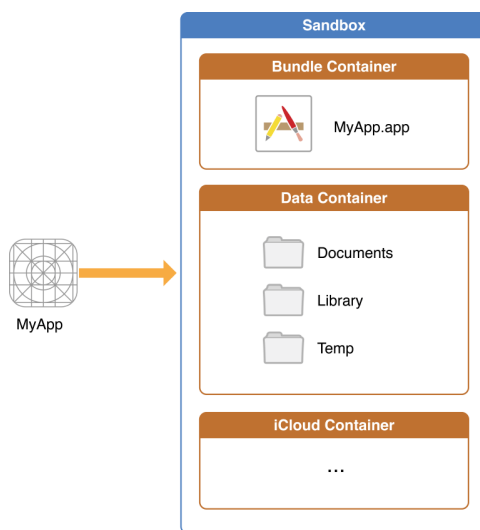


Figura 3.2: Entorno aislado de una aplicación en iOS [11]

Cada aplicación instalada por el usuario se ejecuta en un *entorno aislado*⁴. Como consecuencia de esto, tiene denegado el acceso de archivos guardados por otra aplicación y no puede realizar cambios en el dispositivo. Si una aplicación requiere acceder a información que no es suya, lo puede hacer únicamente usando servicios de iOS. Lo mismo sucede si quiere ejecutar procesos en segundo plano. En la Figura 3.2 se puede observar lo mencionado anteriormente: cada aplicación tiene su directorio *Home* para sus archivos, el cual es otorgado aleatoriamente al momento de la instalación.

El entorno seguro de una aplicación comienza desde el momento de su desarrollo. Los IDE de iOS construyen las aplicaciones utilizando las técnica ASRL⁵. Por ejemplo, Xcode compila automáticamente con ASLR activada. De esta forma, se asegura que todas las regiones de memoria son aleatorias al momento de ejecución [10], reduciendo la probabilidad de muchos *exploits* sofisticados.

⁴Traducción propuesta para el término *sandbox*.

⁵*Address Space Layout Randomization* es una técnica de seguridad informática involucrada en la prevención contra ataques de desbordamiento de *buffer*.

3.2.2. Controles de privacidad

iOS ayuda a evitar que las aplicaciones accedan a la información personal de un usuario sin permiso. Es por ello que, el acceso a ciertos recursos, necesita autorización explícita del usuario. Las aplicaciones pueden solicitar un permiso solamente mientras se esté ejecutando. A su vez, los usuarios pueden optar por no permitir este acceso, y pueden cambiar su elección en cualquier momento. Cabe aclarar que una aplicación puede utilizar un recurso sólo si se le ha dado permiso.

Los permisos restringen el acceso a:

- | | |
|-----------------------------|------------------|
| ■ Servicios de Localización | ■ Micrófono |
| ■ Contactos | ■ Cámara |
| ■ Calendarios | ■ Salud |
| ■ Recordatorios | ■ <i>HomeKit</i> |
| ■ Fotos | ■ Redes Sociales |
| ■ Compartir Bluetooth | |

De querer otorgar o revocar algún permiso, el usuario debe ir a la configuración de privacidad (**Ajustes/Privacidad**). En la Figura 3.3 se observa una captura de pantalla de dicha configuración.

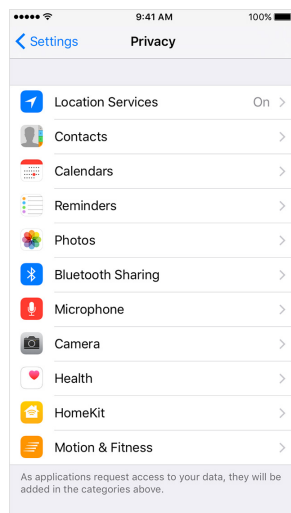


Figura 3.3: Control de privacidad de iOS 9.

Capítulo 4

Análisis Comparativo

Android e iOS son dos plataformas muy populares entre los dispositivos móviles. Es por ello, que existes muchas muchas formas de comparar sus respectivos módulos de seguridad.

La medida de comparación propuesta en [28] consiste en analizar la seguridad de una aplicación móvil en cada fase del ciclo de vida. Como resultado del análisis, afirma que la seguridad en los dispositivos móviles no es responsabilidad exclusiva del desarrollador. Es responsabilidad de todos los usuarios y las personas participan desde la fase de desarrollo hasta la fase de ejecución del usuario final. También depende de cómo se usa la aplicación en el dispositivo y de cómo es almacenada en las tiendas de aplicaciones.

Por otro lado, en [21] el enfoque es distinto. Dada aplicaciones que se ejecutan en Android e iOS, con una funcionalidad casi idéntica, compara los permisos requeridos en cada plataforma al momento de instalarla. El análisis mostró que las aplicaciones en iOS tienden a utilizar más permisos en comparación con sus contrapartes en Android, y tienen más probabilidades de acceder a recursos confidenciales que pueden causar vulnerabilidades de privacidad.

En el campo de los métodos formales, [20, 15, 26] proponen una especificación formal que describe el módulo de seguridad de Android. Dicha especificación está desarrollada con el asistente de pruebas Coq, que es utilizado tanto en la especificación como en la demostración formal de diferentes propiedades sobre el modelo de seguridad representado.

Por otra parte, en [27], el enfoque se centra en distintos ataques al modelo de seguridad de iOS desde la perspectiva de un agente de seguridad intentando recuperar información de un dispositivo. Como resultado de

ese análisis, se presenta un *workflow* que analiza los diferentes aspectos de los sistemas de cifrado de iOS. Dicho *workflow* se presenta como una herramienta que ayudan al oficial de seguridad a obtener información crítica que no está disponible dentro de las herramientas de configuración estándar de iOS.

Finalmente, en este capítulo se propone una forma de comparar distinta a las anteriores. Consiste en analizar distintas características presentes en ambas plataformas, poniendo foco en los permisos que se pueden modificar *en tiempo de ejecución*. El análisis está basado en documentos oficiales de seguridad, tales como [5, 6, 10]. Al final del capítulo se presenta una crítica sobre las funcionalidades mencionadas en el análisis.

4.1. Analizando Android

4.1.1. Autenticación del usuario

Android provee diversas formas para que un usuario se autentique, con el objetivo de desbloquear la pantalla. Desde los comienzos, la autenticación se realizaba mediante el PIN, contraseña y patrones. A partir de la versión 5.0, se introduce el concepto llamado *TrustAgents*, el cuál permite mecanismos de desbloqueo más flexibles, tales como:

- Reconocimiento facial.
- Un determinado lugar, configurado a través de Google Maps.
- Reconocimiento de voz.
- Ciertos dispositivos, tales como el auto (a través de Bluetooth).

La novedad en la versión 6.0 es que soporta el lector de huellas digitales.

Dependiendo del método utilizado para autenticarse, el sistema operativo provee dos componentes: *Gatekeeper* y *Fingerprint*. El primer componente realiza la autenticación del patrón/contraseña del dispositivo en un entorno de ejecución de confianza TEE¹; mientras que el segundo componente es el encargado de verificar que la huella detectada por el sensor es válida. Estos componentes interactúan con el *Keystore*² para soportar el uso de *tokens*

¹El *Trust Execution Environment* es una zona segura del procesador principal en la cual se provee una ejecución segura e íntegra, tanto de código fuente como de datos. El TEE aísla por *hardware* el acceso a cierta memoria y provee mecanismos de I/O para dicha memoria [19].

²Es un componente para almacenar las claves criptográficas, el cual dificulta su extracción, ya que asegura dos cosas: una clave nunca entra en una aplicación y una clave nunca sale de una zona segura [8].

de autenticación respaldados por *hardware* (*AuthTokens*).

La verificación del desbloqueo de la pantalla ocurre en el TEE, tal como se observa en la Figura 4.1.

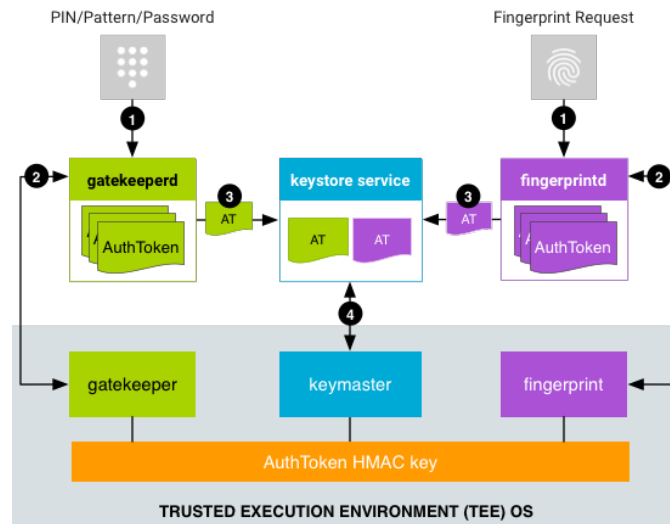


Figura 4.1: Proceso de autenticación en Android[5].

Como consecuencia de esta forma de autenticarse, se destacan las siguientes ventajas:

- Al permitir el desbloqueo con datos biométricos, se acelera y se simplifica el proceso de autenticación. Los usuarios eligen este sistema de desbloqueo en un 91 % [6].
- Al realizarse la autenticación en un TEE, se mejora la protección contra ataques de fuerza bruta, ya que se incrementa exponencialmente el tiempo de espera para el desbloqueo [6].

Estas mejoras permiten que los desarrolladores de aplicaciones tengan más opciones de seguridad para sus datos y sus comunicaciones.

4.1.2. Seguro desde el arranque

Android ofrece la funcionalidad de garantizar un arranque seguro del dispositivo, comenzando desde un lugar confiable del *hardware* hasta que se monta la partición. Durante el arranque, el sistema operativo verifica que la versión de Android no se haya alterado respecto a la de fábrica, informando mediante alertas en caso contrario y ofreciendo opciones para resolverlo.

Dependiendo de la implementación de la funcionalidad, el sistema operativo puede ofrecer una acción al usuario o evitar el arranque hasta que se haya solucionado el problema [5].

En la figura 4.2 se observa el diagrama de flujo del *Arranque Seguro*³, el cual termina en cuatro estados posibles:

- Verde: indica que se pudo verificar correctamente el arranque del sistema.
- Amarillo: indica que se pudo validar el certificado correspondiente a la partición de arranque. Requiere la huella dactilar para continuar el inicio.
- Naranja: indica que el dispositivo pudo ser modificado, ya que no se pudo verificar la partición de arranque. Requiere acción del usuario para continuar.
- Rojo: indica que falló la verificación. Es decir, no pudo validar ninguna partición.

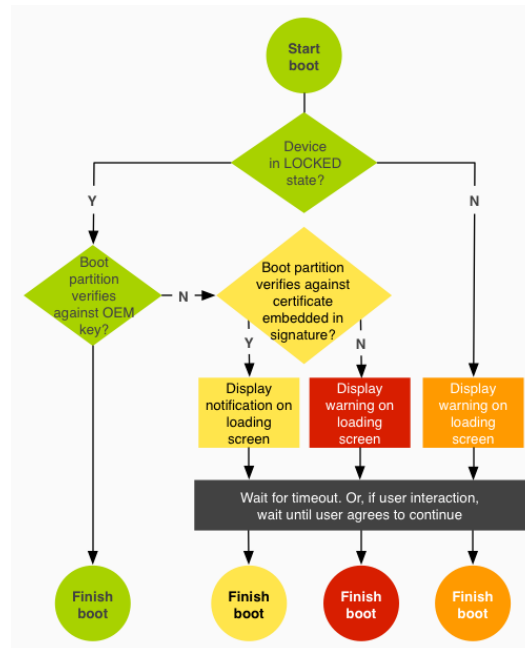


Figura 4.2: Diagrama de flujo del *Arranque seguro* [6].

³Traducción propuesta del término *Verified Boot*.

4.1.3. Cifrado de la partición de datos

Mediante una opción de la configuración, Android permite cifrar todos los datos de usuario presentes en un dispositivo, utilizando claves de cifrado simétricas. Una vez cifrado, el proceso es transparente para el usuario. Es decir, todos los datos creados se cifran antes de enviarlos al disco y todas las lecturas descifran los datos antes de devolverlos al proceso que realizó la llamada.

Esta funcionalidad se activa desde **Ajustes/Seguridad/Cifrar Teléfono**, como se observa en la Figura 4.3. Al activarse dicha opción, se cifran los datos privados de las aplicaciones, el contenido de la tarjeta SD y los datos personales, pudiendo cambiar más adelante el alcance de los componentes afectados por el cifrado. Mientras esté activa dicha opción, cada vez que arranque el dispositivo, el usuario debe proporcionar sus credenciales para poder acceder a cualquier parte del disco.

La primera vez que apareció esta funcionalidad fue en la versión 3.0. Sin embargo, a partir de la versión 5.0 fue fuertemente recomendado a los fabricantes de dispositivos que habiliten esta característica. La novedad incorporada en Android Marshmallow es que se puede cifrar el contenido de la tarjeta SD, permitiendo que sea ilegible si es removida del dispositivo.

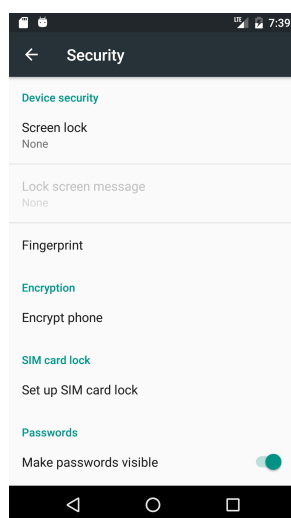


Figura 4.3: Captura de Ajustes/Seguridad.

4.1.4. Permisos

Debido a que cada aplicación Android opera en un *entorno aislado*⁴, las aplicaciones deben compartir de manera explícita recursos y datos. El camino utilizado para realizar dicho intercambio es la declaración de permisos, tal como se introdujo en la sección 2.3.1. El presente informe se centra en los permisos *Normales* y *Peligrosos*; cómo se otorgan y cómo se deniegan.

En las versiones anteriores a Android Marshmallow, al prepararse para instalar una aplicación, el sistema operativo mostraba un diálogo al usuario indicando los permisos solicitados y se le consultaba si deseaba continuar con la instalación. En caso afirmativo, el sistema otorgaba todos los permisos solicitados e instalaba la aplicación. En el caso contrario, no se instalaba la aplicación. El usuario quedaba preso si quería instalar una aplicación: no podía otorgar o denegar permisos individuales; debía otorgar o denegar todos los permisos solicitados como un bloque. Una vez concedidos, los permisos seguían vigentes mientras la aplicación estuviera instalada. Solo se eliminaban si se desinstala dicha aplicación.

A partir de la versión 6.0 se propone un nuevo modelo de permisos, donde los usuarios pueden administrar en tiempo de ejecución los permisos *peligrosos* requeridos por una aplicación. En este modelo, los permisos se agrupan para facilitar el control de la privacidad de los usuarios.

Dichos grupos son:

- *Almacenamiento*: Regula el acceso al almacenamiento externo⁵, permitiendo la lectura o la escritura desde el mismo.
- *Calendario*: Permite leer, modificar o eliminar los calendarios del usuario. Incluye además el manejo de los eventos presentes en un calendario.
- *Cámara*: Regula el acceso de la cámara del dispositivo, permitiendo capturar imágenes y grabar videos.
- *Contactos*: Permite leer, modificar o eliminar los contactos presentes en el dispositivo.
- *Localización*: Regula el acceso a la ubicación del dispositivo, ya sea la ubicación precisa (GPS) o la ubicación aproximada (WIFI/Móvil).
- *Mensajes SMS*: Permite escribir mensajes SMS, leerlos o eliminarlos. Además, permite interceptar los mensajes entrantes.

⁴Ver sección Entorno aislado para cada aplicación.

⁵En Android, cuando se habla de almacenamiento externo, se refiere a la tarjeta SD.

- *Microfono*: Regula el acceso al micrófono del dispositivo, permitiendo además grabar el sonido obtenido. No se incluyen en este grupo los permisos para capturar sonidos provenientes de una llamada telefónica.
- *Teléfono*: Regula el acceso a la información relacionada a una llamada telefónica, tales como iniciar una llamada, obtener datos de una llamada en curso, manipular el log de llamadas, entre otros.
- *Sensores*: Permite acceder a los datos de los sensores del dispositivo. Se incluyen el giroscopio, el acelerómetro, sensor del ritmo cardíaco, entre otros.

En contraposición a lo que ocurría en las versiones previas, en Android Marshmallow durante la instalación de una aplicación se le otorgan todos los permisos *normales* y ningún permiso *peligroso*. Como consecuencia de esto, cada vez que una aplicación necesita acceder a un recurso protegido por un permiso *peligroso*, tiene que solicitarlo en tiempo de ejecución. Por ejemplo, si una aplicación requiere leer un contacto, la primera vez aparece una notificación pidiendo autorización explícita al usuario, tal como se observa en la Figura 4.4a. El usuario puede otorgar el permiso, denegarlo una vez o denegarlo para siempre. Si elige la última opción, no volverá a aparecer la notificación solicitando dicho permiso. Dichas opciones se encuentran en **Ajustes/Aplicaciones/{nombre de la aplicación}/Permisos**. En la Figura 4.4b se observa una captura de las configuraciones de los permisos para una aplicación móvil.

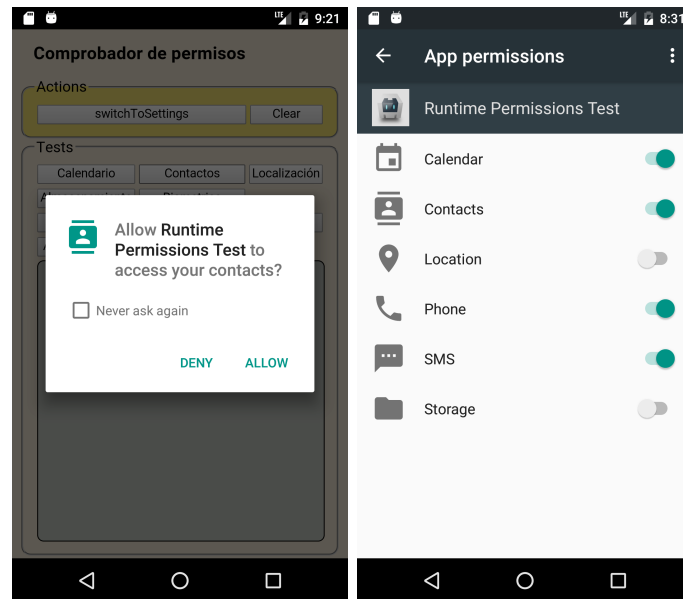
Debido a que el usuario puede revocar algunos los permisos en cualquier momento, una aplicación debe comprobar si dispone de ellos cada vez que se ejecuta. De lo contrario no va a poder acceder a las funciones de las cuales no tiene permiso. Sin embargo, el usuario no puede revocar los permisos *normales*; solamente se eliminan al desinstalar la aplicación.

4.2. Analizando iOS

4.2.1. Bloqueo del dispositivo

Al configurar el código de desbloqueo para un dispositivo, el usuario activa la protección de datos automáticamente. El sistema admite códigos alfanuméricos de cuatro dígitos, de seis dígitos y de longitud arbitraria, salvo que el dispositivo tenga un lector de huellas; en ese último caso deberá contar con al menos seis dígitos.

Además de desbloquear el dispositivo, el código provee entropía a ciertas claves de cifrado del sistema. El hecho de que esté muy ligado con el UID, añade una seguridad extra: no se puede intentar quebrar dicho código



(a) Solicitud de un permiso en tiempo de ejecución. (b) Descripción general de los permisos otorgados.

Figura 4.4: Permisos en Android Marshmallow.

fuera del dispositivo. Es por ello que cuanto más seguro sea el código de desbloqueo, más segura será la clave de cifrado.

A fin de desalentar los posibles ataques de fuerza bruta, se generan retardos cada vez mayores tras la introducción de un código inválido en la pantalla de bloqueo. Los retardos están calibrados suponiendo que la frecuencia entre un ataque y otro es de 80 milisegundos [10]. En dispositivos que cuentan con un *Secure Enclave*, los retardos se aplican mediante dicho componente. Si el dispositivo se reinicia durante un tiempo de demora, la demora aún se aplica, con el temporizador empezando de nuevo para el periodo actual.

4.2.2. Arranque seguro

Cada dispositivo es seguro desde el arranque, ya que la arquitectura del sistema fue pensada para integrar *hardware*, *software* y servicios, con el objetivo de obtener seguridad a lo largo de todos los componentes que conforman el núcleo del sistema [10]. Cuando se prende un dispositivo cuyo sistema operativo es iOS, se siguen los siguientes pasos para asegurar la integridad del arranque del sistema⁶:

⁶Traducción propuesta para el término *Boot Chain*

1. Se ejecuta el código alojado en la *BootROM*. Dicho código es inmutable y seguro por definición.
2. Se verifica la firma del *bootloader* LLB⁷, ya que fue certificado por Apple con la clave pública *Apple Root CA*. La misma está alojada en la *BootROM*.
3. Pasada la validación, empieza la carga del kernel de iOS mediante *iBoot*.
4. El siguiente paso es asegurar la integridad del software. Los dispositivos con un procesador A7 o superior, cuentan con un coprocesador llamado *Secure Enclave*⁸ para verificar dicha integridad.

Si fallan alguno de estos pasos, el dispositivo entra en *Modo de Recuperación*⁹ y se lo debe conectar a iTunes via USB para restaurar la configuración de fábrica.

4.2.3. Cifrado de archivos

Cada vez que se guarda un archivo en la partición de datos, el sistema operativo crea una clave AES-256 para ése archivo. Dicha clave es única y es utilizada por el *Secure Enclave* para cifrar el archivo. Luego, ésa clave se empaqueta con una (o varias) de las claves de clases, dependiendo de la accesibilidad que va a tener el archivo [10].

Cada vez que se abre un archivo, ocurre el proceso inverso: su metadata es desempaquetada con la clave del sistema de archivos, revelando la clave particular del archivo y las clases que lo protegen. Se vuelve a desempaquetar, esta vez, con las claves de las clases, para finalmente descifrar al archivo con su clave única. El proceso completo se observa en la Figura 4.5.

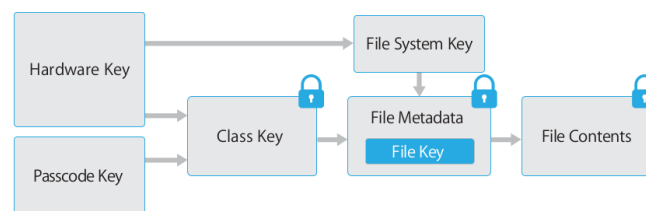


Figura 4.5: Proceso para descifrar un archivo [10].

⁷Low Level Bootloader, por sus siglas en inglés

⁸*Secure Enclave* es un coprocesador con memoria cifrada e incluye generación de números aleatorios por hardware [10].

⁹Traducción del término *recovery mode*.

4.2.4. Firma de código de las aplicaciones

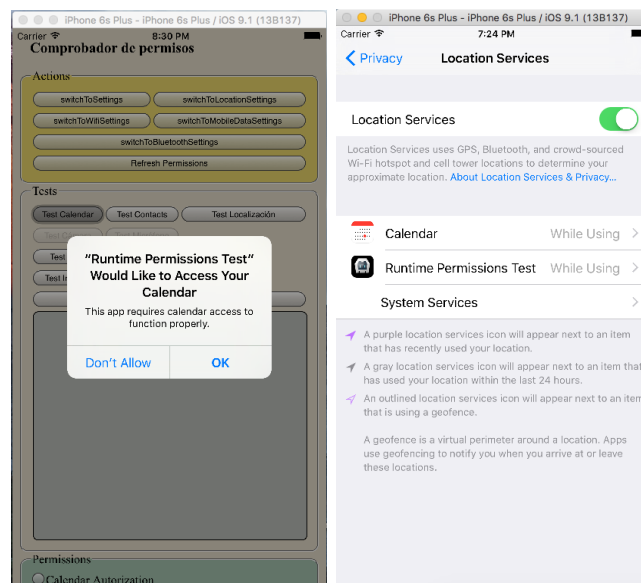
iOS garantiza que todas las aplicaciones proceden de una fuente conocida y aprobada. Por ello, requiere que todo el código ejecutable se firme con un certificado emitido por Apple. La firma de código obligatoria extiende el concepto de cadena de confianza del sistema operativo a las aplicaciones e impide que aplicaciones de terceros carguen código sin firmar o utilicen código que se modifique automáticamente.

Para poder desarrollar aplicaciones en dispositivos iOS, los desarrolladores deben registrarse. Apple verifica la identidad real de cada desarrollador, ya sea una persona individual o una empresa, antes de emitir su certificado. Este certificado permite a los desarrolladores firmar sus aplicaciones y enviarlas a la tienda para su distribución. Además, todas las aplicaciones de la tienda han sido revisadas para garantizar que funcionan según lo esperado y que no contienen errores ni otros problemas evidentes.

A diferencia de otras plataformas móviles, iOS no permite a los usuarios instalar aplicaciones procedentes de sitios web que no estén firmadas, ni ejecutar código que no sea de confianza [10]. Durante la ejecución de una aplicación se comprueba la firma de código de todas las páginas de la memoria ejecutable a medida que se cargan para garantizar que una aplicación no se ha modificado desde la última vez que se instaló o actualizó [10].

4.2.5. Permisos

Los controles de privacidad en iOS restringen el acceso de las aplicaciones a la información personal del usuario. El principal control es el sistema de permisos, el cual es encargado de custodiar el acceso a ciertos recursos. Al momento de instalar una aplicación en el dispositivo, no le se otorga ningún permiso. Es responsabilidad de cada aplicación solicitar los permisos que requiere al momento de utilizar el recurso custodiado. Por ejemplo, en la Figura 4.6a se observa el pedido del permiso al componente Calendario. Sin la autorización explícita del usuario, dicha aplicación tiene restringido el acceso a ese recurso.



(a) Captura del pedido de un permiso. (b) Aplicaciones que requirieron un permiso.

Figura 4.6: Permisos en iOS 9.

El usuario puede modificar la configuración de privacidad desde **Ajustes/Privacidad**. Se pueden aprobar o denegar cualquier permiso que haya sido solicitado. Aún más, se puede denegar “para siempre”: está pensado para evitar que aparezca constantemente una notificación solicitando determinado permiso. En la Figura 4.6b se observan las aplicaciones que requirieron el permiso *Localización*.

Los permisos restringen el acceso a:

- *Localización*: Permite determinar aproximadamente la ubicación del dispositivo. Utiliza una combinación de cierta información del celular, WiFi, Bluetooth y GPS para determinarla.
- *Contactos*: Regula el acceso a los contactos del dispositivo, ya sea para crearlos, modificarlos o eliminarlos.
- *Calendarios*: Regula el acceso a los calendarios, incluyendo las citas y eventos incluidos en él.
- *Recordatorios*: Permite leer, modificar o eliminar un recordatorio. Los recordatorios son pequeñas notas, listas de tareas, entre otras cosas.
- *Fotos*: Regula el acceso a la galería de imágenes de la cámara. También tiene la capacidad de crear un álbum de fotos dentro de la aplicación de fotos.

- *Compartir a través de Bluetooth*: Regula cuáles aplicaciones pueden compartir datos a través de Bluetooth.
- *Micrófono*: Regula el acceso al micrófono.
- *Cámara*: Regula el acceso a la cámara del dispositivo.
- *Salud*: Regula el acceso a la información de salud y estado físico del usuario, tanto las que recopiló el dispositivo como las insertadas por el usuario.
- *HomeKit*: Regula el acceso a los accesorios hogareños registrados en el dispositivo.
- *Redes Sociales*: Permite realizar actividades relacionadas a una red social, tales como crear una sesión de red, obtener el feed de actividad para un usuario, hacer una nueva entrada.
- *Diagnóstico*: Regula cuáles datos de diagnóstico sobre el dispositivo se envían a Apple. Esos datos incluyen información sobre el rendimiento del sistema, capacidad restante en el dispositivo, entre otras.
- *Publicidad*: Permite inhabilitar los anuncios basados en intereses.

4.3. Crítica

Teniendo presente el análisis realizado en las secciones anteriores, se realiza a continuación una crítica sobre los modelos de seguridad de ambas plataformas. La misma está compuesta por cuatro niveles gradualmente más complejos que engloban aspectos de la seguridad que van desde que se prende el dispositivo hasta el uso de una aplicación.

4.3.1. Arranque verificado

Desde la primera versión, iOS ofrece la funcionalidad de verificación desde el arranque del dispositivo para ver si fue modificado respecto de la versión de fábrica. Por lo tanto, dicha verificación es transparente para el usuario. En caso de fallar, el dispositivo entra en *modo de recuperación*, teniendo que conectarlo a una computadora para restaurar la configuración de fábrica.

En Android, el arranque verificado se agregó en la versión 4.4 pero recién en la versión 6.0 se hizo obligatorio para los fabricantes. La principal ventaja respecto de iOS es que tiene cuatro estados finales de verificación en vez de dos. Se identifican por colores: verde, amarillo, naranja y rojo. Los estados verde y rojo son equivalentes a los estados de iOS: válido e

inválido, respectivamente. Los estados amarillo y naranja corresponden a una validación parcial¹⁰. Android deja librado a la decisión del usuario para continuar o suspender el arranque.

4.3.2. Cifrado del sistema de archivos

Tanto Android con iOS proveen la funcionalidad de cifrar los datos alojados en el sistema de archivos. Sin embargo, tienen una política distinta al momento de aplicarlo: en iOS está siempre habilitada y no se puede deshabilitar; mientras que en Android, se deja esta decisión al usuario.

En Android, al soportar almacenamiento externo, se corre el riesgo de que se acceda a los datos por fuera del dispositivo. Es por ello que, a partir de la versión 6.0, el usuario puede cifrar la tarjeta SD. De esta manera, reduce el riesgo mencionado.

Por su parte, iOS está libre del riesgo mencionado en el párrafo anterior, ya que todo su almacenamiento es interno. Además, iOS agrega una medida extra de seguridad: los archivos se cifran utilizando varias claves, entre las que se encuentran el código de desbloqueo y UID¹¹. Gracias a ello, se dificulta muchísimo la tarea de quebrar el cifrado por fuera del dispositivo.

4.3.3. Bloqueo del dispositivo

Android cuenta con cinco métodos de bloqueo de pantalla: PIN, patrón, contraseña, desbloqueo facial y huella digital. Por otro lado, iOS dispone de sólo dos métodos: PIN y huella digital.

Independientemente del método utilizado para bloquear la pantalla, tanto en Android como en iOS, el proceso de desbloqueo ocurre en un entorno seguro (TEE y *Secure Enclave*, respectivamente). Como consecuencia de esto, las claves se procesan siempre allí y no son conocidas por el resto de los componentes. Además, se incrementa el tiempo de espera del bloqueo, dificultando un ataque por fuerza bruta.

iOS agrega una medida de seguridad extra: el código de desbloqueo está muy ligado al UID. Es por ello que no se puede realizar ataques al código de desbloqueo fuera del dispositivo.

¹⁰Ver apartado 4.1.2.

¹¹Clave única por dispositivo. Más datos en la sección 3.1.

4.3.4. Seguridad de las aplicaciones

Inicialmente, tanto Android como iOS tienen un enfoque similar, en el sentido de que ambos se apoyan en sus propias tiendas de aplicaciones en los que comprueban la seguridad de las miles de aplicaciones que se encuentran a disposición del usuario. Sin embargo, tienen una diferencia filosófica respecto de los proveedores de aplicaciones: en iOS todas las aplicaciones tienen que descargarse de la tienda oficial. Es más, para poder subir una aplicación, su desarrollador pasa por un proceso muy estricto de registro. En cambio, al ser un sistema más abierto, Android favorece la instalación de aplicaciones de terceros, ya que deja abierta la posibilidad de instalar aplicaciones “sueltas” e incluso tiendas diferentes a Play Store. Es peligroso, pero le da una flexibilidad inmensa, frente a la rigidez de iOS. Esta libertad, sin embargo, tiene un precio, y es la presencia de *malware* disfrazado de aplicaciones legítimas.

Respecto a la ejecución de una aplicación, ambos sistemas aíslan los procesos en ejecución en un *entorno aislado*¹². Como consecuencia de esto, se destacan dos resultados: se evita que una aplicación pueda tomar control de todo el sistema; y se impiden también que una aplicación conozca datos de otra.

Al comparar la gestión de permisos de ambas plataformas, encontramos varias similitudes. Lo primera cosa en común es que a las aplicaciones no se le otorgan permisos al momento de instalación. Otra cosa que comparten es que si una aplicación necesita algún permiso, debe requerirlo mientras se ejecuta, pudiendo el usuario otorgarlo o denegarlo. La última similitud es que, desde la configuración de privacidad, el usuario puede revocar u otorgar permisos a las aplicaciones.

Respecto a cómo se definen los permisos, se observan diferencias de concepto. En Android están orientados según el riesgo implícito al otorgarlos. En cambio, en iOS los permisos están orientados a los componentes. Sin embargo, se pueden comparar según los componentes que son afectados por un permiso. De esta manera, se puede saber cuáles componentes son protegidos por permisos en ambas plataformas y cuáles están presentes solamente en una de ellas. Con estos datos, se confeccionó la Tabla 4.1.

Respecto del alcance del sistema de permisos, se observó una falta de granularidad de los permisos que se pueden modificar *en tiempo de ejecución*. En Android, un permiso es a nivel de grupo. Por lo tanto, el usuario otorga o deniega para todo el grupo. Por ejemplo, si se pide permisos del grupo *Contactos* se obtienen los permisos para leer, modificar

¹²Traducción propuesta para el término *sandbox*.

Permisos		
<i>Ambas plataformas</i>	<i>Solo en Android</i>	<i>Solo en iOS</i>
Calendario	-	-
Contactos	-	-
Cámara	-	-
Localización	-	-
-	-	Compartir por Bluetooth
Micrófono	-	-
-	Teléfono	-
Sensores	-	-
-	SMS	-
-	Almacenamiento	-
-	-	<i>Homekit</i>
-	-	Redes Sociales
-	-	Diagnóstico
-	-	Publicidad

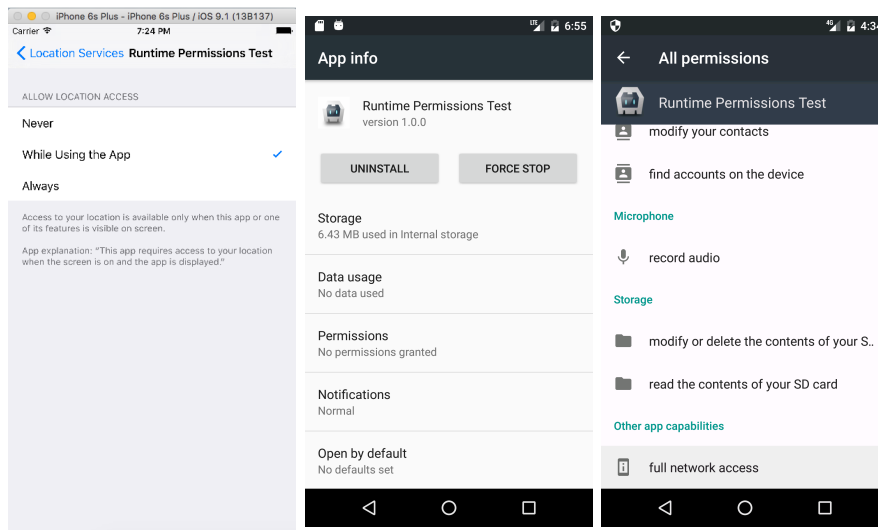
Cuadro 4.1: Comparación de permisos entre Android 6.0 e iOS

o eliminar los contactos. Para una aplicación que administra los contactos tiene sentido tener todos los permisos, pero para una aplicación de acceso a una red social, no tiene sentido que pueda eliminar un contacto. La misma situación ocurre en iOS: se otorga un permiso de acceso a todas las funcionalidades de un determinado componente. Como consecuencia de ello, el usuario está delegando a una aplicación demasiados permisos y no tiene expresividad para decir que funcionalidades autoriza.

Otra cosa a destacar es la cobertura del sistema de permisos. Las dos plataformas dejan funcionalidades principales del dispositivo sin permisos modificables *en tiempo de ejecución*. En Android, los permisos *normales* son otorgados al instalarse la aplicación y no pueden ser quitados. Entre ellos se encuentran: Acceso a Internet, Compartir vía Bluetooth, Información del dispositivo, entre otros. Además, no hay un permiso para proteger los datos de las Redes Sociales. Por su parte, iOS también deja algunos componentes sin permisos modificables *en tiempo de ejecución*. Entre ellos se encuentran: Acceso a Internet, SMS, entre otros. Tampoco tiene la suficiente granularidad para administrar el acceso a los datos de las llamadas telefónicas.

Para finalizar analizamos cómo se muestran al usuario los permisos adquiridos por una aplicación. En iOS solamente se listan los permisos requeridos por una aplicación, como se observa en la Figura 4.7a. En cambio, en Android es confuso cómo se informa. En la Figura 4.7b se visualizan los

datos de una aplicación. Si se observa el apartado *Permisos*, la leyenda indica que no fue otorgado ningún permiso *peligroso*. Ingresando en ese apartado, queda claro lo mencionado anteriormente. Sin embargo, nada dice sobre los demás permisos. Si se presiona sobre los tres puntos, se accede a todos los permisos, como se observa en la Figura 4.7c. Ahí nace la confusión: primero indica que “No hay permisos otorgados”, pero indagando un poco, se descubre que fueron otorgados todos los permisos *normales*.



(a) Permisos requeridos por una aplicación. (b) La aplicación no tiene ningún permiso. (c) Tiene todos los permisos *normales*.

Capítulo 5

Apache Cordova

Apache Cordova es un *framework* para el desarrollo de aplicaciones móviles creado originalmente por Nitobi [25]. Adobe Systems lo compró en 2011, y lo renombró como PhoneGap. Posteriormente, lanzó una versión de código abierto llamada Apache Cordova [16].

El *framework* permite a los programadores crear aplicaciones para dispositivos móviles utilizando HTML5, CSS3, y JavaScript, con el objetivo de lograr un desarrollo multiplataforma. Las aplicaciones se ejecutan dentro de *contenedores*¹ para cada plataforma y dependen de APIs estándar para acceder a las funcionalidades de los dispositivos [16]. De esta forma, se evita el desarrollo en el lenguaje nativo de cada plataforma. Las aplicaciones resultantes son híbridas, lo que significa que no son realmente aplicaciones nativas ni tampoco basadas en la web. La mezcla de fragmentos de código nativos e híbridos ha sido posible desde la versión 1.9.

A lo largo de este capítulo se introduce el *framework* Apache Cordova: su arquitectura y el paradigma de programación asociado a él.

5.0.5. Arquitectura

Una aplicación desarrollada con Apache Cordova está compuesta por varios componentes. Dichos componentes son:

- *WebView*: Es un componente nativo que incorpora contenido web a una aplicación. En algunas plataformas, puede proporcionar a la aplicación toda su interfaz de usuario.
- *Aplicación Web*: La aplicación en sí se implementa como una página web. Por defecto, se utiliza un archivo local llamado `index.html`, que hace referencia a CSS, JavaScript y otros recursos que sean necesarios.

¹Traducción propuesta para el término *wrapper*.

La aplicación se ejecuta en un *WebView* dentro del contenedor de aplicaciones nativo.

- *Plugins*: Proporcionan una interfaz para comunicarse entre Apache Cordova y los componentes nativos. Esto permite invocar código nativo desde JavaScript.

En la Figura 5.1 se observa el diagrama de los componentes principales.

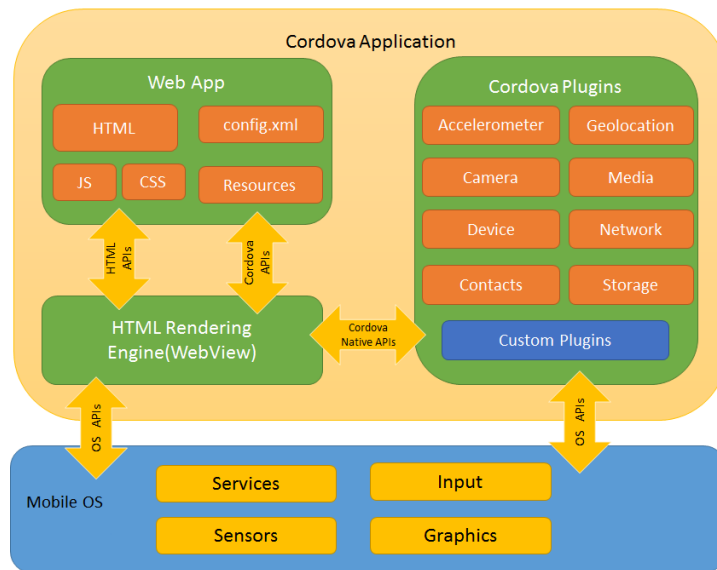


Figura 5.1: Arquitectura de Apache Cordova [16].

5.0.6. Aplicaciones Nativas vs Aplicaciones Híbridas

Decimos que una aplicación es nativa cuando fue desarrollada específicamente para un sistema operativo móvil (Objective-C o Swift para iOS, Java para Android).

Dado que una aplicación nativa se desarrolla siguiendo las recomendaciones técnicas del sistema operativo, no solo tiene la ventaja de tener un rendimiento más rápido sino que también es más amigable, ya que tiene una apariencia similar a la mayoría de las otras aplicaciones nativas del dispositivo. Además, pueden acceder y utilizar fácilmente las capacidades integradas del dispositivo del usuario.

Por otra parte, decimos que una aplicación es híbrida cuando está basada en la web, construida principalmente con HTML5 y JavaScript, que luego se ejecuta en un *contenedor* que proporciona acceso a las características de

la plataforma nativa.

Las aplicaciones híbridas se ven como una aplicación nativa, pero, fuera del marco básico de la aplicación, sus contenidos son provistos por un servidor externo. Cuando el usuario instala una aplicación híbrida, se descargan la mayor parte del contenido. Posteriormente, mientras el usuario navega por la aplicación, se van descargando el resto de los contenidos según se vayan necesitando. Entre sus ventajas principales se encuentran que: dispone de todos los *frameworks* disponibles para lenguajes web, el desarrollo es multiplataforma, y se programa en un solo lenguaje.

5.0.7. Plugins

Los *Plugins* son una parte esencial dentro del ecosistema de Apache Cordova. Cada uno de ellos es un paquete de código inyectado que permite que la aplicación móvil se comuniquen con la plataforma nativa en la que se ejecuta. Los *plugins* brindan acceso a funcionalidades del dispositivo y de la plataforma que normalmente no están disponibles para las aplicaciones basadas en la web [17].

Para hacer efectivo el mencionado acceso, los *plugins* exponen una única interfaz de JavaScript. Apache Cordova realiza el enlace entre la interfaz Javascript y las correspondientes bibliotecas de códigos nativos, las cuales son provistas por el mismo *plugin*. En esencia, un *plugin* oculta las diversas implementaciones de código nativo detrás de una interfaz de JavaScript común.

Por otra parte, podemos clasificar a los *plugins* en dos grandes grupos: *Core Plugins* y *Plugins Personalizados*. Los primeros proporcionan acceso a las funcionalidades básicas del dispositivo, tales como la batería, la cámara, los contactos, entre otros. Apache Cordova es el responsable de mantenerlos. Mientras que los segundos, son desarrollados por terceros y proporcionan accesos adicionales a funciones que no están necesariamente disponibles en todas las plataformas. El autor de dicho *plugin* es responsable mantenerlo y de documentar cuáles plataformas soporta. Apache Cordova proporciona un buscador de *plugins* o también se pueden descargar por **npm**.

Al crear un proyecto en Apache Cordova, por defecto no tiene *plugins* instalados. Si se desean algún *plugin*, inclusive los *Core Plugins*, deben ser agregados explícitamente.

Capítulo 6

Hacia un Framework Comparativo

Android e iOS permiten cambiar ciertos permisos de una aplicación luego de haberla instalado en el dispositivo. En este contexto, se ha desarrollado un *framework* para determinar empíricamente el alcance de dichos cambios en los sistemas de permisos de ambas plataformas.

El *framework* es una aplicación móvil y está compuesto por varios tests. Cada test pone a prueba a un componente del dispositivo, permitiendo así conocer el alcance de los permisos correspondientes a dicho componente. De esta manera, se busca dejar en evidencia posibles vulnerabilidades presentes en los modelos de seguridad. Adicionalmente, se pone especial énfasis en la relación existente entre la privacidad del usuario y el sistema de permisos, analizando cuál es la cobertura del sistema respecto de los datos sensibles para la privacidad.

En las siguientes secciones se detallarán los distintos tests que componen el *framework*. Además, se mencionarán las conclusiones arribadas luego de correr los tests mencionados anteriormente.

6.1. Ejecución de los test

Durante el momento de desarrollo de aplicaciones móviles multiplafoma, un desarrollador necesita tener los dispositivos reales para probar y depurar sus aplicaciones, hasta alcanzar el funcionamiento deseado. Sin embargo, a veces ocurre que el desarrollador no cuenta con todos los dispositivos en los que se ejecuta su aplicación. Es por ello que tanto Android como iOS proveen emuladores oficiales para salvar dicho inconveniente.

Los emuladores permiten interactuar de la misma manera que se haría

con un dispositivo real, pero con el ratón y el teclado, y mediante los botones y los controles del emulador. De forma dinámica, permiten acercar y alejar la imagen, cambiar la orientación e incluso tomar capturas de pantalla. Además, son compatibles con pantallas táctiles y botones de *hardware* virtuales, incluyendo también las operaciones con dos dedos.

Para el presente trabajo se decidió utilizar los emuladores oficiales para testear el *framework* propuesto. A continuación se mencionan las principales características de los emuladores.

6.1.1. Emulador oficial de Android

Android provee un emulador para que un desarrollador pueda simular un dispositivo en la computadora de desarrollo. El emulador es compatible con teléfonos y tablets Android, y con dispositivos Android Wear y Android TV [7].

Cuando una aplicación se ejecuta en el emulador puede usar los servicios de la plataforma Android para invocar otras aplicaciones, acceder a la red, almacenar y recuperar datos, y mostrar notificaciones. El emulador tiene controles que permiten enviar mensajes de texto y administrar llamadas telefónicas con facilidad, especificar la ubicación del dispositivo, simular lecturas de huellas digitales, y simular las propiedades de la batería.

El emulador requiere una configuración de un AVD¹ para determinar la apariencia, funcionalidad y versión del sistema del dispositivo simulado. Los AVD permiten definir determinados aspectos de *hardware* de los dispositivos emulados. Cuentan con tipos de dispositivos predefinidos, como el teléfono Nexus, y permiten crear definiciones propias. En la Figura 6.1 se observa una captura del ADV utilizado para el desarrollo del *framework*.

6.1.2. Simulador oficial de iOS

Apple provee un simulador, disponible dentro de Xcode, que se ejecuta como una aplicación independiente. El simulador presenta la interfaz de usuario de un iPhone, iPad, Apple Watch o Apple TV como una ventana en el escritorio de su Mac y le permite manipular esta pantalla y la aplicación que se ejecuta [12].

El simulador proporciona la capacidad de emular muchas combinaciones diferentes de tipo de dispositivo y versión de sistema operativo. Un tipo de dispositivo es un modelo de iPhone, iPad o Apple TV. Cada combinación de dispositivo-sistema operativo tiene su propio entorno de simulación con sus propias configuraciones y aplicaciones. También puede agregar simuladores

¹Android Virtual Device, por sus siglas en inglés.

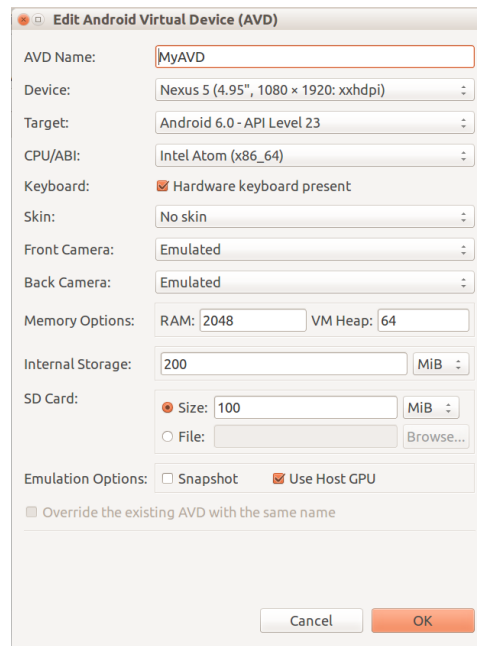


Figura 6.1: Configuración de un AVD.

para una combinación específica que quiera probar. Sin embargo, no todas las combinaciones de tipo de dispositivo y versión de SO están disponibles. Dicha configuración se realiza desde **Hardware/Dispositivo**.

Para probar el funcionamiento de una aplicación, un desarrollador puede ejecutar su aplicación en el simulador directamente desde el proyecto Xcode. El simulador proporciona herramientas para ayudar en la depuración de sus aplicaciones. Por ejemplo, se puede simular la ubicación dentro de una aplicación. En la Figura 6.2 se observa el simulador emulando un iPhone 5 (iOS 8.0).

A pesar de todas las funcionalidades que provee, el simulador tiene sus limitaciones. No permite instalar aplicaciones desde App Store. Sumado a esto, muchas de las aplicaciones que vienen preinstaladas en dispositivos iOS no están disponibles en el simulador [12]. Además, no todos los errores y problemas de rendimiento pueden detectarse mediante pruebas en el simulador.

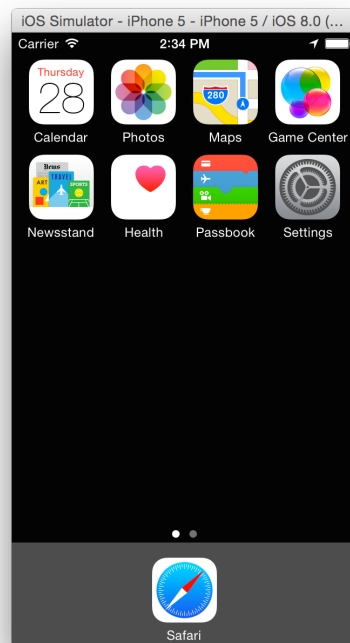


Figura 6.2: Pantalla de inicio del simulador de iOS [12].

6.2. Vista principal

Al iniciar la aplicación se observan dos áreas principales: Acciones y Test, como se puede observar en la Figura 6.3.

La primera área contiene un botón para acceder a la configuración de los permisos del dispositivo. Allí, el *tester* puede cambiar manualmente los permisos requeridos por la aplicación. Además, se encuentra un botón para limpiar la consola del *framework*.

La segunda área se subdivide en dos: la parte de los tests y la parte de la consola. Una parte contiene un conjunto de botones que, al presionarse, ejecutan un test. El resultado se imprimirá en la consola con tipografía color verde si fue exitoso; en cambio, se imprimirá con tipografía color roja de si falla.

A continuación se mencionan los componentes que se pueden testear con el *framework*:

- Contactos
- Calendario

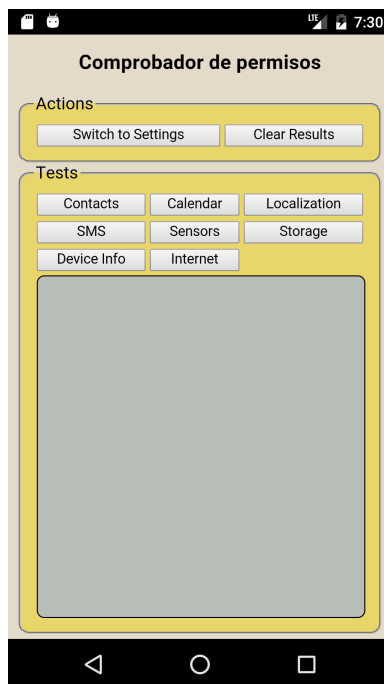


Figura 6.3: Áreas del *framework*.

- Geolocalización
- SMS
- Sensores
- Almacenamiento
- Información del dispositivo
- Acceso a Internet

6.2.1. Funciones no compatibles

El emulador oficial de Android es compatible con la mayoría de las funciones de un dispositivo, pero no incluye la posibilidad de virtualizar los siguientes componentes [7]:

- WiFi
- Bluetooth
- NFC²

²Del ingles *Near Field Communication*. Es una tecnología de comunicación inalámbrica, de corto alcance y alta frecuencia que permite el intercambio de datos entre dispositivos.

- Manipulación de la tarjeta SD
- Conexión USB
- Micrófono
- Cámara

Al no poder manipular la tarjeta SD, no es posible testear las funcionalidades multimedia: no se puede grabar audio, ni video ni sacar fotos. Por lo tanto, no se agregaron al *framework* tests para los componentes listados anteriormente.

6.3. Catálogo de Tests

En esta sección se listarán todos los test que conforman el *framework*. Para cada uno de ellos se detalla el algoritmo, los plugins de Apache Cordova que se utilizaron para desarrollarlo y una serie de capturas que muestran los casos exitosos y fallidos.

Para acceder al panel de configuraciones, se utilizó el *plugin* cordova.plugins.diagnostic (v. 3.0.4) de Apache Cordova.

De ahora en adelante, cuando se diga ‘consola’, se refiere a la consola del *framework*.

6.3.1. Contactos

Algoritmo 1 Test de Contactos.

- 1: Se imprimen por consola todos los contactos.
 - 2: Se crea un nuevo contacto.
 - 3: Se vuelven a imprimir por consola todos los contactos.
-

El test consiste en crear un contacto y luego listar todos los contactos presentes en el dispositivo. En caso de ser exitoso, se imprimen los contactos por consola. De lo contrario, se imprime un error. En la Figura 6.4a se observa el resultado del test cuando no se tiene el permiso correspondiente; mientras que en la Figura 6.4b se observa el caso exitoso.

Para desarrollarlo, se utilizó el *plugin* cordova-plugin-contacts (v. 2.3.1) de Apache Cordova.

Finalmente, para correr el test, es necesario tener el permiso **Contacto**, tanto para Android como para iOS.



(a) Sin permisos.

(b) Con permisos.

Figura 6.4: Testeando la administración de los contactos.

6.3.2. Calendario

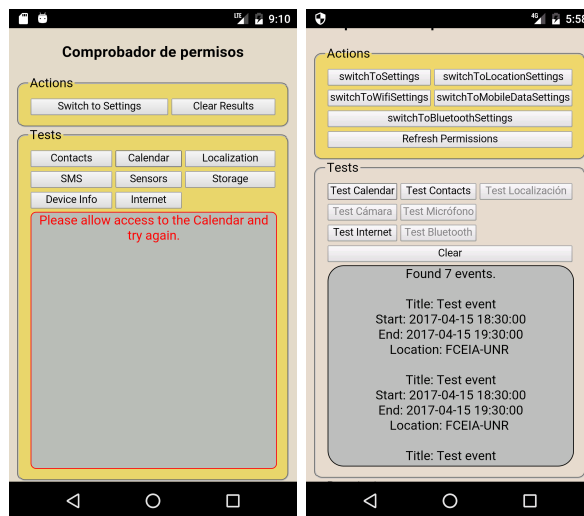
Algoritmo 2 Test del Calendario.

- 1: Se crean las fechas *startDate* y *endDate*.
 - 2: Se crea un evento que empieza en la fecha *startDate* y termina en la fecha *endDate*.
 - 3: Se imprimen por consola los eventos entre las fechas *startDate* y *endDate*.
-

El test consiste en crear un evento en un determinado rango de fechas y luego listar todos los eventos dentro del rango. En caso de ser exitoso, se muestran los datos del evento. De lo contrario, se muestra un error. En la Figura 6.5a se observa el resultado del test cuando no se tiene el permiso correspondiente; mientras que en la Figura 6.5b se observa el caso exitoso.

Para desarrollarlo, se utilizó el *plugin* cordova-plugin-calendar (v. 4.6) de Apache Cordova.

Finalmente, para correr el test, es necesario tener el permiso **Calendario** para Android. En cambio, para iOS, es necesario tener el permiso **Recordatorios**.



(a) Sin permisos.

(b) Con permisos.

Figura 6.5: Testeando la administración del calendario.

6.3.3. Geolocalización

Algoritmo 3 Test de Geolocalización.

- 1: Se censa el GPS.
 - 2: Se imprimen por consola los datos.
-

El objetivo del presente test es obtener los datos de la ubicación actual. En caso de tener los permisos correspondientes, obtiene tanto la ubicación precisa (GPS) como la aproximada (WIFI/Móvil). En la Figura 6.6a se observa el resultado del test cuando no se tiene el permiso correspondiente; mientras que en la Figura 6.6b se observa el caso exitoso.

Para desarrollarlo, se utilizó el *plugin* cordova-plugin-geolocation (v. 2.4.3) de Apache Cordova.

Al momento de realizar las pruebas, se configuró el emulador de Android para que simule las coordenadas (-122° , 37°), tal como se observa en la Figura 6.7. Dicha configuración también se realizó en emulador oficial de iOS.

Finalmente, para correr el test, es necesario tener el permiso *Localización*, tanto para Android como para iOS.



(a) Sin permisos.

(b) Con permisos.

Figura 6.6: Testeando la geolocalización.

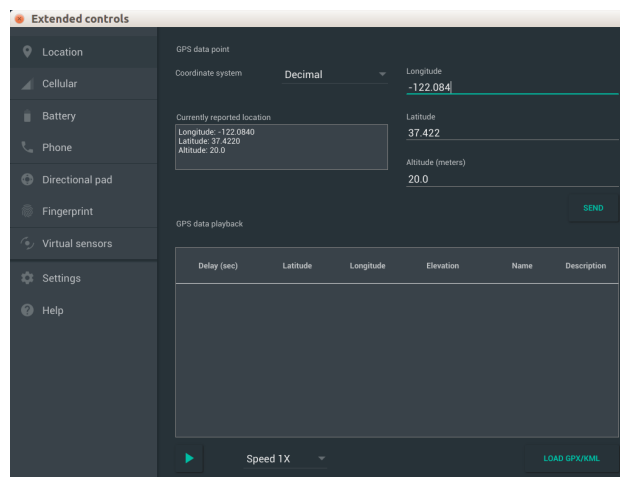


Figura 6.7: Panel de configuración del emulador de Android.

6.3.4. SMS

Algoritmo 4 Test de SMS.

- 1: Se envía un SMS de prueba.
 - 2: Se imprime por consola el resultado del test.
-

El test consiste en enviar un mensaje SMS. En un principio, se diseñó con la capacidad de enviar, leer y recibir mensajes. Al momento de desarrollarlo, se encontró una restricción de seguridad en iOS: a partir de la version 8 no se pueden acceder a los mensajes SMS desde una aplicación instalada por el usuario [13, 14]. En cambio, en Android sí se pueden acceder, siempre que se tengan el permiso correspondiente. Dicho permiso es **SMS**.

Como consecuencia de lo mencionado en el párrafo anterior, se decidió no implementar las funcionalidades incompatibles, quedando en el test la posibilidad de enviar mensajes SMS. Para desarrollarlo, se utilizó el *plugin* cordova-plugin-sms (v. 0.1.11) de Apache Cordova.

En la Figura 6.8a se observa el resultado del test cuando no se tiene el permiso correspondiente; mientras que en la Figura 6.8b se observa el caso exitoso. Finalmente, para correr el test, es necesario tener el permiso **SMS** para Android. Sin embargo, no es necesario tener permisos para correr el test en iOS.



(a) Sin permisos.

(b) Con permisos.

Figura 6.8: Testeando los mensajes SMS.

6.3.5. Almacenamiento

Algoritmo 5 Test de Almacenamiento.

- 1: Se realiza una captura de la pantalla.
 - 2: Se guarda la captura en el dispositivo.
 - 3: Se imprime por consola el resultado del test.
-

El presente test fue diseñado para probar el alcance de los permisos de escritura sobre el sistema de archivos que tiene cada plataforma. En la Figura 6.9a se observa el resultado del test cuando no se tiene el permiso correspondiente; mientras que en la Figura 6.9b se observa el caso exitoso.

Para desarrollar el presente test se utilizó el *plugin* cordova-screenshot (v. 0.1.5) de Apache Cordova.

Finalmente, para correr el test, es necesario tener el permiso **Almacenamiento** para Android. Sin embargo, no es necesario tener permisos para correr el test en iOS.

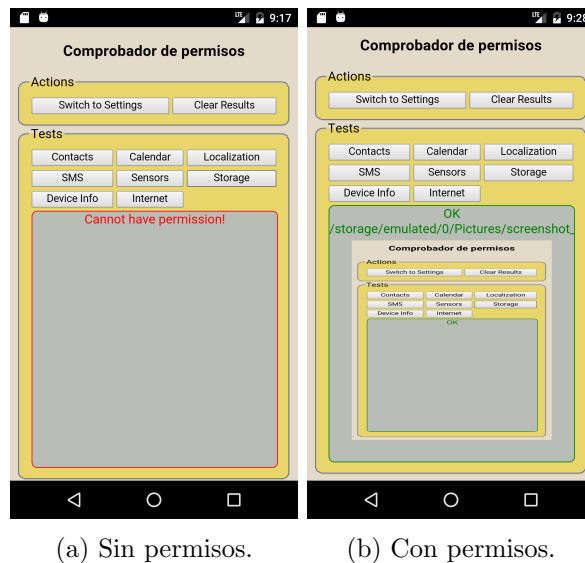


Figura 6.9: Testeando el almacenamiento del dispositivo.

6.3.6. Información del Dispositivo

El objetivo de este test es obtener datos del dispositivo donde corre la aplicación. Entre otros datos, se obtiene la plataforma, modelo del dispositivo y su número de serie. En la Figura 6.10 se observan los datos

Algoritmo 6 Test de Información del Dispositivo.

- 1: Se obtienen los datos del dispositivo.
 - 2: Se imprime por consola los datos obtenidos.
-

obtenidos.

Para desarrollarlo se utilizó el *plugin* cordova-plugin-device (v.1.1.6) de Apache Cordova.

Cabe aclarar que no fueron necesarios permisos en ninguna de las dos plataformas para poder correrlo.

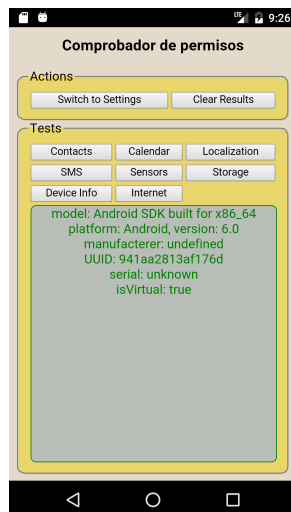


Figura 6.10: Testeando Información del Dispositivo.

6.3.7. Sensores

Algoritmo 7 Test de los Sensores.

- 1: Se inicializa un **timer** con 5 **seg** para detener las mediciones.
 - 2: Se inicia la medición del acelerómetro.
 - 3: Se inicia la medición del giroscopio.
 - 4: Se muestran los resultados en la consola.
-

El objetivo de este test es obtener datos de dos sensores del dispositivo: acelerómetro y giroscopio. Para ello, se configura un **timer**. Durante el tiempo que este activo, se tomarán distintos muestreos; y cuando ocurra el *timeout* se mostrarán los datos en la consola de la aplicación. En la Figura

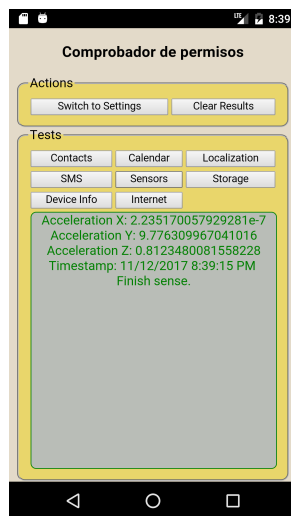


Figura 6.11: Testeando los sensores.

6.11 se observan los datos obtenidos.

Para desarrollar el presente test se utilizaron los *plugins* cordova-plugin-device-motion y cordova-plugin-gyroscope, de Apache Cordova.

Cabe aclarar que no fueron necesarios permisos en ninguna de las dos plataformas para poder correrlo.

6.3.8. Internet

Algoritmo 8 Test de conexión a Internet.

- 1: Se realiza una consulta GET HTTP hacia logo del DCC.
 - 2: Se decodifica la imagen (viene codificada en Base64).
 - 3: Se imprime por consola un `tag `, cuyo `source` es el dato decodificado.
 - 4: Se realiza una consulta POST HTTP hacia `httpbin`.
 - 5: Se imprime por consola la respuesta.
-

El objetivo del presente test es establecer una comunicación a través de Internet. Al momento de desarrollarlo, no se requirió de ningún *plugin* de Apache Cordova.

En la Figura 6.12a se observa el resultado del test cuando no se tiene conexión a Internet; mientras que en la Figura 6.12b se observa el caso de establecer una comunicación por Internet.



(a) Sin conexión a Internet. (b) Con conexión a Internet.

Figura 6.12: Testeando el acceso a Internet.

Cabe aclarar que, al ejecutarse el framework desde un emulador, para probar el acceso a Internet se habilitó/deshabilitó la Red Inalámbrica de la computadora donde se corrieron los emuladores.

Por último, no son necesarios permisos en ninguna de las dos plataformas para poder utilizar el presente test.

6.4. Resultados experimentales

El *framework* desarrollado en este trabajo tiene la capacidad de testear los componentes *Contactos*, *Calendario*, *Geolocalización*, *SMS*, *Sensores*, *Almacenamiento*, *Información del dispositivo* y *Acceso a Internet*, tal como se explicó en la Sección 6.2.

Un primer resultado es poder clasificar los componentes testeados en cuatro clases, según requieran autorización del usuario para utilizarlos. Dichas clases son:

- Clase A: componentes que requieren autorización explícita en ambas plataformas para poder utilizar las funcionalidades que proveen;
- Clase B: componentes que requieren autorización explícita solamente en Android;

- Clase C: componentes que requieren autorización explícita solamente en iOS;
- Clase D: componentes que no requieren autorización explícita para poder utilizar las funcionalidades que proveen.

Las clases son mutuamente excluyentes. Luego de correr los tests provistos por el *framework*, se armó el Cuadro 6.1. Cada test permitió clasificar a un componente presente en ambas plataformas.

A continuación, se detalla como están conformadas las clases.

<i>Permisos</i>			
<i>Clase A</i>	<i>Clase B</i>	<i>Clase C</i>	<i>Clase D</i>
Contactos	-	-	-
Calendario	-	-	-
Geolocalización	-	-	-
-	SMS ³	-	-
-	Almacenamiento	-	-
-	-	-	Sensores
-	-	-	Información del dispositivo
-	-	-	Acceso a Internet

Cuadro 6.1: Clasificación de permisos según si requieren autorización.

6.4.1. Clase A

Los componentes que conforman esta clase son *Contactos*, *Calendario* y *Geolocalización*. La característica común entre ellos es que requieren autorización explícita del usuario para interactuar con ellos. Si el usuario deniega el correspondiente permiso, no se puede acceder a ninguna funcionalidad del componente.

El primer componente a analizar es *Contactos*. En Android, requiere el permiso *peligroso* llamado *Contacto* y en iOS requiere el permiso llamado con el mismo nombre. En ambas plataformas, los dos permisos abarcan las mismas funciones: permiten crear un contacto, borrarlo, editarlo y obtener un listado de todos los contactos presentes en el dispositivo.

Luego, se continúa analizando el componente *Calendario*. En Android permite administrar todo lo relacionado con los eventos: crear un evento, modificarlo y eliminarlo. Además, permite administrar varios calendarios, permitiendo por ejemplo, tener un calendario que contenga los eventos laborales y otro con los eventos personales. En iOS, estas funcionalidades

³Aplica solamente al envío de mensajes.

están separadas en dos componentes: *Calendarios* y *Recordatorios*. Cada uno de ellos tiene su permiso que lo regula. Sin embargo, el test presente en el *framework* prueba solamente el componente *Recordatorios*. Para poder correrlo, es necesario tener el permiso *Recordatorios*. Por otra parte, en Android, el permiso *peligroso* que regula las funcionalidades mencionadas anteriormente, se llama *Calendario*.

A continuación se analiza el último componente de esta clase, llamado *Geolocalización*. En ambas plataformas, provee el acceso a la ubicación del dispositivo, ya sea la ubicación precisa (GPS) o la aproximada (Wi-Fi/Móvil). Tanto en Android como en iOS, el permiso que regula dichas funcionalidades se llama *Localización*.

6.4.2. Clase B

Esta clase está compuesta por los componentes *SMS* y *Almacenamiento*. Ellos tienen en común que para utilizar sus funcionalidades no requieren permisos en iOS. Sin embargo, necesitan autorización explícita en Android.

Se comenzará el análisis por el componente *SMS*. Sus funcionalidades son: enviar un mensaje, eliminarlo, obtener los mensajes entrantes y obtener la lista de todos los mensajes, ya sean recibidos o enviados. Sin embargo, el test presente en el *framework* solamente prueba el envío de mensajes, ya que en iOS, a partir de la versión 8 no permite acceder al resto de las funcionalidades desde una aplicación de terceros (es decir, no nativa) [13, 14]. En cambio, Android permite acceder a todas las funcionalidades mencionadas; y se pueden acceder teniendo el permiso *peligroso* llamado *SMS*.

Finalmente, se hablará sobre el componente *Almacenamiento*. El permiso que regula al componente en Android tiene su mismo nombre. Dicho permiso resguarda las funcionalidades que permiten leer y escribir la tarjeta SD. El test presente en el *framework* solamente prueba escribir en el sistema de archivos.

6.4.3. Clase C

Los componentes que agrupa la presente clase son aquellos que requieren algún permiso en iOS y no requieren permisos en Android para poder utilizar sus funcionalidades.

Cabe aclarar que la presente clase es puramente teórica, ya que ninguno de los componentes analizados pertenecen a ella. Sin embargo, se mantuvo

en el informe ya que es posible que exista algún componente en dicha clase, que pueda ser descubierto en futuras versiones del *framework*.

6.4.4. Clase D

La presente clase agrupa componentes que no requieren autorización explícita del usuario para utilizar las funcionalidades que brindan. Sus miembros son *Información del dispositivo*, *Sensores* y *Acceso a Internet*.

Como se explica en la sección 4.1.4, si una aplicación requiere algún permiso asociado a un componente de esta clase, el sistema operativo se lo otorga al momento de instalación. Es por ello que los miembros de esta clase pueden ser potencialmente peligrosos respecto de la privacidad del usuario, ya que exponen medios o información que, combinadas con otros componentes, pueden violar dicha privacidad.

Por ejemplo, supongamos que se tiene instalada una aplicación maliciosa de administración de contactos. Supongamos también, para simplificar, que requiere solamente de dos componentes: *Contactos* y *Acceso a Internet*. Según el Cuadro 6.1, al usuario se le requeriría explícitamente el permiso *Contactos*. Sin embargo, dicha aplicación maliciosa obtiene el permiso correspondiente a *Acceso a Internet* sin que el usuario sea notificado. Ergo, ¡puede robar todos los contactos!

Empecemos el análisis de los miembros de la clase. El primer componente nos brinda datos relacionados al dispositivo en el cual se está corriendo la aplicación. Especifica el modelo del dispositivo, el cual es establecido por el fabricante del dispositivo y puede ser diferente en todas las versiones del mismo producto. También se puede obtener información relacionada a la plataforma del dispositivo: nombre, versión, quién lo manufacturó, su UUID⁴ y si es emulado o no.

Luego, se continúa con el componente *Sensores*. Dicho componente controla el acceso a los sensores del dispositivo. El test presente en el *framework* recolecta datos de dos sensores: el acelerómetro y el giroscopio. Con los datos obtenidos, se podrían calcular varios movimientos del dispositivo, tales como inclinación, vibración, rotación o balanceo.

El último componente de esta clase es el que nos provee acceso a Internet. En la actualidad, son muchas las aplicaciones móviles que utilizan Internet. Según el ranking confeccionado por [24], las 10 aplicaciones más populares utilizan Internet. El test permite realizar una petición GET y una petición

⁴Del inglés Universally Unique Identifier. Es un numero de 128 bits que identifica al dispositivo biunívocamente.

POST sin notificar al usuario. Esto es potencialmente muy peligroso ya que una aplicación maliciosa puede enviar y recibir datos sin que el usuario lo note. Por ejemplo, supongamos que tenemos una aplicación maliciosa que nos oficia de navegador GPS. Supongamos también que el usuario le otorga el permiso para utilizar el GPS. Sin embargo, dicha aplicación podría enviar a través de Internet todas las rutas realizadas por el usuario, sin que el mismo sea notificado.

Capítulo 7

Conclusiones y Trabajos Futuros

Los dispositivos móviles plantean nuevos problemas de seguridad. Son más propensos a perderse o ser robados, exponiendo datos confidenciales. Cabe afirmar que el *software* que controla la privacidad de miles de millones de personas puede ser considerado *software* crítico. Es por ello que es de vital importancia analizar los modelos de seguridad de Android e iOS, con el objetivo de encontrar fortalezas y debilidades.

Si bien son muchos los trabajos que comparan los modelos de seguridad de ambas plataformas, el presente trabajo realiza aportes basado en un análisis sobre distintas características de seguridad provistas por ambas plataformas con el objetivo de preservar la privacidad del usuario. La principal característica es el sistema de permisos. Android Marshmallow incorpora a una novedosa manera de administrar el sistema de permisos: los permisos *peligrosos* se pueden otorgar o denegar en cualquier momento; mientras que iOS ya tenía una forma similar desde versiones anteriores.

Como primer aporte, en el cuarto capítulo se realizó un análisis comparativo entre algunas características presentes en los modelos de seguridad de ambas plataformas. Ellas son: verificación del arranque del sistema operativo, cifrado del sistema de archivos, bloqueo del dispositivo y sistemas de permisos. Todas ellas se eligieron porque son importantes a la hora de resguardar la privacidad del usuario. Es por ello que la característica analizada con mayor profundidad fue el sistema de permisos. Como fruto del análisis, se logró establecer una medida de comparación entre los permisos presentes en ambas plataformas. La medida propuesta es la siguiente: *dos permisos son similares si resguardan un componente que provee la misma funcionalidad*. Por ejemplo, ambas plataformas tienen un componente que permite obtener la localización del dispositivo. Además, dichos componentes

tienen un permiso, en cada plataforma, que lo resguarda. Por lo tanto, dichos permisos son similares. Utilizando la medida propuesta, todos los permisos que un usuario puede cambiar en tiempo de ejecución quedan clasificados en tres grupos: *Ambas Plataformas*, *Solo en Android* o *Solo en iOS*. Cabe aclarar que los grupos son mutuamente excluyentes. El resultado de la clasificación se observa en el Cuadro 4.1.

Otro aporte del presente informe es el *framework* propuesto en el capítulo anterior. Tiene dos funciones principales: determinar empíricamente los alcances de los sistemas de permisos; y establecer una relación entre los permisos presenten en las dos plataformas. El *framework* está compuesto por una batería de tests, teniendo cada uno de ellos la tarea de probar una funcionalidad provista por Android e iOS. Como resultado de la utilización del *framework* propuesto se determinó una clasificación de permisos. El criterio utilizado para clasificarlos fue el siguiente: *un componente pertenece a una clase según requiera autorización explícita del usuario para utilizarlo*. Utilizando el criterio propuesto, se obtuvieron cuatro clases mutuamente excluyentes:

- Clase A: componentes que requieren autorización explícita en ambas plataformas para poder utilizar las funcionalidades que proveen;
- Clase B: componentes que requieren autorización explícita solamente en Android;
- Clase C: componentes que requieren autorización explícita solamente en iOS;
- Clase D: componentes que no requieren autorización explícita para poder utilizar las funcionalidades que proveen.

Se pueden mencionar varias observaciones. La primera de ellas es la clasificación de varios componentes, no solamente los que permiten cambiar su permiso en tiempo de ejecución. Esto se observa en el Cuadro 6.1. Otra observación es que todas las clases contienen al menos un elemento, salvo la *Clase C*. Sin embargo, se mantuvo en el informe ya que, en futuras versiones del *framework*, es posible que pueda ser descubierto algún miembro de ella. Como última observación, algunos de los miembros de la *Clase D* son importantes a la hora de resguardar la privacidad del usuario. Un ejemplo de esto es el permiso de *Acceso a Internet*. Si bien solo no vulnera la privacidad, combinado con algún otro permiso deja expuesta información sensible. Es por ello que algunos miembros de la *Clase D* deberían pertenecer a la *Clase A*.

Para finalizar, se mencionan los trabajos a futuro. Una alternativa posible es incrementar las capacidades del *framework* presentado en el

capítulo anterior. Al ser desarrollado en la plataforma Apache Cordova, se puede extender fácilmente. La información relacionada al desarrollo del *framework* y una guía de cómo extenderlo, se encuentran en el Anexo.

A continuación, se enumeran algunas líneas a seguir:

- Probar y mejorar los test que actualmente conforman el framework en dispositivos reales.
- Desarrollar tests para las funcionalidades que no pueden ser emuladas (ver Sección 5.1.1, [13, 14]).
- Desarrollar un test para poder comparar el cifrado de archivos.
- Android otorga todos los permisos *normales*, tal como se enuncia en la Sección 3.1.4. Pero no sabemos qué permisos otorga iOS. Se podrían desarrollar varios test para descubrirlos.
- Dado que salieron al mercado las versiones Android 7.0 e iOS 10, se podría analizar extender el *framework* para las características de seguridad adicionales en dichas versiones.

Anexo

En este capítulo es de utilidad para aquellos que pretender modificar o extender el *framework* propuesto en este informe. El proyecto se puede clonar desde el repositorio de **GitHub** runtime-permissions-test. Al ser una aplicación híbrida que utiliza *plugins* de Apache Cordova, es requisito indispensable instalarlo.

A continuación se presentan los pasos para instalar el proyecto, arrancado desde cero; y una guía para agregar tests.

Pasos para instalar el proyecto

1. Instalar Apache Cordova.

```
user $> npm install -g cordova
```

2. Clonar el proyecto runtime-permissions-test.

```
user $> https://github.com/rgaluppo/  
runtime_permissions_test.git
```

3. Agregar plataforma según corresponda.

```
user $> cordova platform add ios
```

```
user $> cordova platform add android
```

4. Agregar *plugins* de Apache Cordova.

```
user $> cordova plugin add cordova.plugins.diagnostic@3.0  
user $> cordova plugin add cordova-plugin-calendar  
user $> cordova plugin add cordova-plugin-contacts  
user $> cordova plugin add cordova-plugin-geolocation  
user $> cordova plugin add cordova-sms-plugin  
user $> cordova plugin add https://github.com/gitawego/  
cordova-screenshot.git  
user $> cordova plugin add cordova-plugin-device  
user $> cordova plugin add cordova-plugin-device-motion  
user $> cordova plugin add cordova-plugin-gyroscope@0.1.4
```

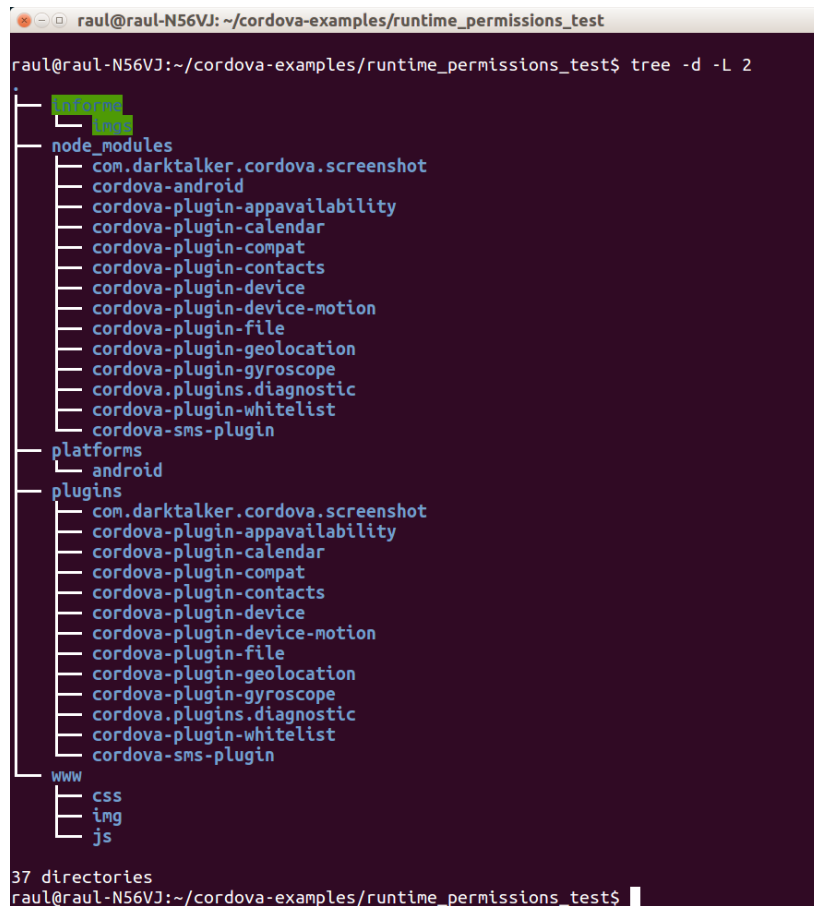
5. Correr la máquina virtual según la plataforma elegida.

```
user $> cordova emulate ios
```

```
user $> cordova emulate android
```

Extensión del Framework

El *framework* propuesto en el capítulo 6 es una aplicación híbrida, desarrollada utilizando Apache Cordova. Para acceder a funciones del teléfono, hay que agregar distintos *plugins*. Algunos de ellos fueron desarrollados por Apache Cordova, mientras que el resto fueron desarrollados por usuarios. Independientemente de su creador, un *plugin* puede descargarse a través de *npm* o un repositorio *git*.



```
raul@raul-N56VJ: ~/cordova-examples/runtime_permissions_test
raul@raul-N56VJ:~/cordova-examples/runtime_permissions_test$ tree -d -L 2
.
├── Informe
├── plugins
├── node_modules
│   ├── com.darktalker.cordova.screenshot
│   ├── cordova-android
│   ├── cordova-plugin-appavailability
│   ├── cordova-plugin-calendar
│   ├── cordova-plugin-compatible
│   ├── cordova-plugin-contacts
│   ├── cordova-plugin-device
│   ├── cordova-plugin-device-motion
│   ├── cordova-plugin-file
│   ├── cordova-plugin-geolocation
│   ├── cordova-plugin-gyroscope
│   ├── cordova.plugins.diagnostic
│   ├── cordova-plugin-whitelist
│   └── cordova-sms-plugin
├── platforms
│   └── android
├── plugins
│   ├── com.darktalker.cordova.screenshot
│   ├── cordova-plugin-appavailability
│   ├── cordova-plugin-calendar
│   ├── cordova-plugin-compatible
│   ├── cordova-plugin-contacts
│   ├── cordova-plugin-device
│   ├── cordova-plugin-device-motion
│   ├── cordova-plugin-file
│   ├── cordova-plugin-geolocation
│   ├── cordova-plugin-gyroscope
│   ├── cordova.plugins.diagnostic
│   ├── cordova-plugin-whitelist
│   └── cordova-sms-plugin
├── www
│   ├── css
│   ├── img
│   └── js
└── 37 directories
raul@raul-N56VJ:~/cordova-examples/runtime_permissions_test$
```

Figura 1: Estructura de archivos de un proyecto Apache Cordova.

Cada proyecto de Apache Cordova tiene una estructura particular, como se observa en la Figura 1. Se pueden destacar tres carpetas:

- **platforms:** contiene las plataformas agregadas al proyecto.
- **plugins:** contiene los *plugins* agregados al proyecto.
- **www:** contiene todos los recursos que utiliza la aplicación híbrida.

Al ser el *framework* una aplicación híbrida, dentro de la carpeta **www** se encuentra una estructura típica de una página web. En ella se encuentra el archivo **index.html**, junto con los recursos necesarios. En caso de requerirse alguna librería de Javascript, debería estar dentro de esta carpeta.

Para el caso particular del *framework* propuesto, se creó la estructura de archivos que se observa en la Figura 2. En el archivo **index.html** se encuentra el diseño de la aplicación. En la carpeta *js* se encuentran varios *scripts*. Hay uno responsable de interactuar con el usuario, mientras que el resto de los *scripts* contiene un test concreto, los cuales fueron explicados en el capítulo 6.

```

raul@raul-N56VJ: ~/cordova-examples/runtime_permissions_test/www
raul@raul-N56VJ:~/cordova-examples/runtime_permissions_test/www$ tree
.
├── css
│   └── index.css
├── img
│   ├── logo.png
│   └── old-logo.png
├── index.html
└── js
    ├── index.js
    ├── jquery-2.1.3.min.js
    ├── testAppAvailability.js
    ├── testCalendar.js
    ├── testCamera.js
    ├── testContacts.js
    ├── testDeviceInfo.js
    ├── testHealth.js
    ├── testInternet.js
    ├── testLocation.js
    ├── testMicrophone.js
    ├── testSensors.js
    ├── testSMS.js
    ├── testStorage.js
    └── utilities.js

3 directories, 19 files
raul@raul-N56VJ:~/cordova-examples/runtime_permissions_test/www$

```

Figura 2: Estructura típica de una aplicación híbrida.

A continuación, se detallarán una serie de pasos que servirán de guía a la hora de extender el *framework*:

1. Pensar un test.

2. Determinar el componente a testear.
3. Buscar *plugins* de Cordova que permitan acceder a las funcionalidades del componente deseado.
4. Desarrollar un *script* de JavaScript que implemente la lógica del test, realizando invocaciones a los *plugins*.
5. Agregar al archivo `index.html` un gesto para interactuar con el usuario (un botón, por ejemplo).
6. Implementar la interacción deseada en el archivo `index.js`.

Bibliografía

- [1] ALLIANCE, O. H. *Open Handset Alliance*. <http://www.openhandsetalliance.com>. Último acceso 4 de Diciembre del 2017.
- [2] ANDROID. *Android 6.0 Marshmallow*. <https://www.android.com/intl/en/versions/marshmallow-6-0/>. Último acceso 4 de Diciembre del 2017.
- [3] ANDROID. *Android Open Source Project*. <https://source.android.com/index.html>. Último acceso 4 de Diciembre del 2017.
- [4] ANDROID. *Android Open Source Project: Licenses*. <https://source.android.com/source/licenses.html>. Último acceso 4 de Diciembre del 2017.
- [5] ANDROID. *Android Open Source Project: Security*. <https://source.android.com/security/index.html>. Último acceso 4 de Diciembre del 2017.
- [6] ANDROID. *Android Security 2015 Year In Review*. https://source.android.com/security/reports/Google_Android_Security_2015_Report_Final.pdf, 2016. Último acceso 4 de Diciembre del 2017.
- [7] ANDROID, D. *Developer Android: Emulador oficial*. <https://developer.android.com/studio/run/emulator.html>. Último acceso 4 de Diciembre del 2017.
- [8] ANDROID, D. *Developer Android: KeyStore System*. <https://developer.android.com/training/articles/keystore.html>. Último acceso 4 de Diciembre del 2017.
- [9] APPBRAIN. *Google Play stats*. <http://www.appbrain.com/stats>. Último acceso 4 de Diciembre del 2017.
- [10] APPLE. *Apple iOS: Security Guide*. <https://www.apple.com/business/docs/iOSSecurityGuide.pdf>. Último acceso 4 de Diciembre del 2017.

- [11] APPLE, D. *iOS Developer Library: File System Programming Guide*. <https://developer.apple.com/library/ios/documentation/FileManager/Conceptual/FileSystemProgrammingGuide/FileSystemOverview/FileSystemOverview.html>. Último acceso 4 de Diciembre del 2017.
- [12] APPLE, D. *iOS Developer: Simulator User Guide*. https://developer.apple.com/library/content/documentation/IDEs/Conceptual/iOS_Simulator_Guide/Introduction/Introduction.html. Último acceso 4 de Diciembre del 2017.
- [13] APPLE, F. D. *Forum Developer Apple: How to listen for sms reception in ios 8?* <https://forums.developer.apple.com/thread/16685>. Último acceso 4 de Diciembre del 2017.
- [14] APPLE, F. D. *Forum Developer Apple: How to read user's sms in an application?* <https://forums.developer.apple.com/thread/22447>. Último acceso 4 de Diciembre del 2017.
- [15] BETARTE, G., CAMPO, J. D., LUNA, C. D., AND ROMANO, A. Verifying android's permission model. In *ICTAC 2015* (2015), LNCS 9399, pp. 485–504.
- [16] CORDOVA, A. *Apache Cordova: Overview*. <https://cordova.apache.org/docs/en/latest/guide/overview/index.html>. Último acceso 4 de Diciembre del 2017.
- [17] CORDOVA, A. *Apache Cordova: Plugins*. <https://cordova.apache.org/docs/en/latest/guide/hybrid/plugins/index.html>. Último acceso 4 de Diciembre del 2017.
- [18] GARTNER. *Worldwide Sales of Smartphones Grew 9 Percent in First Quarter of 2017*. <https://www.gartner.com/newsroom/id/3725117>. Último acceso 4 de Diciembre del 2017.
- [19] GLOBALPLATFORM. *The Trusted Execution Environment - White Paper*. http://www.globalplatform.org/documents/GlobalPlatform_TEE_White_Paper_Feb2011.pdf, 2011. Último acceso 4 de Diciembre del 2017.
- [20] GOROSTIAGA, F. Especificación e implementación de un prototipo certificado del sistema de permisos de android. Tesina de grado, Licenciatura en Ciencias de la Computación, Universidad Nacional de Rosario, Argentina, Octubre 2016.
- [21] HAN, J., YAN, Q., GAO, D., ZHOU, J., AND DENG, H. R. Comparing mobile privacy protection through cross-platform applications. *Network & Distributed System Security Symposium (NDSS)* (2013).

- [22] HAN, J., YAN, Q., GAO, D., ZHOU, J., AND DENG, H. R. Android or ios for better privacy protection? *International Conference on Secure Knowledge Mangagement in Big-data era (SKM)* (2014).
- [23] iTUNES APP STORE. *App Store Metrics*. <http://www.pocketgamer.biz/metrics/app-store/app-count/>. Último acceso 4 de Diciembre del 2017.
- [24] OF APPS, B. *App Download and Usage Statistics 2017*. <http://www.businessofapps.com/data/app-statistics/>. Último acceso 4 de Diciembre del 2017.
- [25] PHONEGAP, A. *PhoneGap Beliefs, Goals, and Philosophy*. <https://phonegap.com/blog/2012/05/09/phonegap-beliefs-goals-and-philosophy/>. Último acceso 4 de Diciembre del 2017.
- [26] ROMANO, A. Descripción y análisis formal del modelo de seguridad de android. Tesina de grado, Licenciatura en Ciencias de la Computación, Universidad Nacional de Rosario, Argentina, Junio 2014.
- [27] TEUFL, P., ZEFFERER, T., STROMBERGER, C., AND HECHENBLAIKNER, C. ios encryption systems: Deploying ios devices in security-critical environments. In *Security and Cryptography (SECRYPT), 2013 International Conference on* (2013), IEEE, pp. 1–13.
- [28] YOGITA CHITTORIA, N. A. Application security in android-os vs ios. *International Journal of Advanced Research in Computer Science and Software Engineering* 4 (2014).